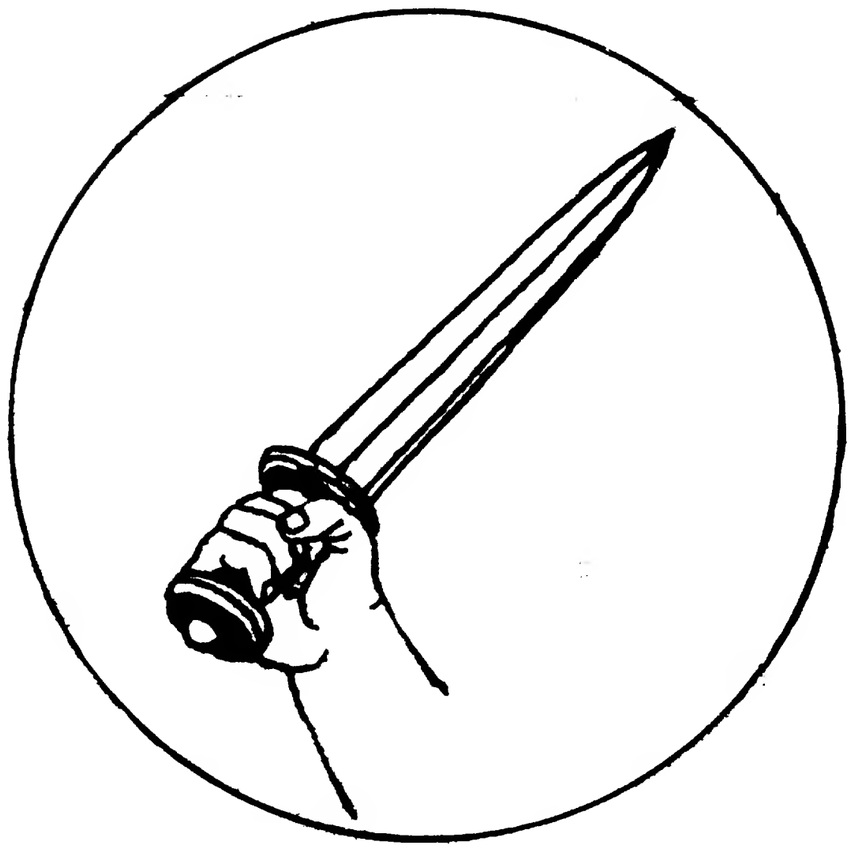
AI-TR-419

TRUTH MAINTENANCE SYSTEMS

FOR PROBLEM SOLVING



Jon Doyle

January 1978

This blank page was inserted to preserve pagination.

AI-TR-419

Truth Maintenance Systems for Problem Solving

by

Jon Doyle

Massachusetts Institute of Technology

January 1978

This report is a revised version of a dissertation submitted to the Department of Electrical Engineering and Computer Science of the Massachusetts Institute of Technology on May 12, 1977 in partial fulfillment of the requirements for the degree of Master of Science.

This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number [redacted].

Abstract:

The thesis developed here is that reasoning programs which take care to record the logical justifications for program beliefs can apply several powerful, but simple, domain-independent algorithms to

{1} maintain the consistency of program beliefs,

{2} realize substantial search efficiencies, and

{3} automatically summarize explanations of program beliefs.

These algorithms use the recorded justifications to maintain the consistency and well-founded basis of the set of beliefs. The set of beliefs can be efficiently updated in an incremental manner when hypotheses are retracted and when new information is discovered. The recorded justifications also enable the pinpointing of exactly those assumptions which support any particular belief. The ability to pinpoint the underlying assumptions is the basis for an extremely powerful domain-independent backtracking method. This method, called Dependency-Directed Backtracking, offers vastly improved performance over traditional backtracking algorithms.

These techniques of recording and using justifications also indicate methods for structuring the deductive process so that the justification-derived arguments for certain types of deductions can be automatically summarized. The levels of detail in a hierarchical problem solver can be separated from each other by this summarization. The separation is accomplished by replacing a set of beliefs at one level by the higher-level beliefs from which they derive. This is important in improving the coherence of explanations, and in further improving the search efficiency of dependency-directed backtracking. Modest extensions of this method are useful in automatically generalizing the results of certain forms of computations.

This report describes techniques for representing, recording, maintaining and using justifications for beliefs. In addition, we present an annotated implementation of a domain-independent program making these functions easily available to programs in a wide range of applications.

Thesis Supervisor: Gerald Jay Sussman
Title: Esther and Harold E. Edgerton Associate Professor
of Electrical Engineering and Computer Science

To my parents,

Leo Michael Boyle
and
Marilyn Catherine Miller Boyle

Acknowledgements

I owe much to many for their attention and cooperation during this research. Many of the ideas in this report are due, directly or indirectly, to Gerald Jay Sussman and Richard M. Stallman. Much of my progress on these problems has derived from the patient advice, inspiring encouragement, and close cooperation of Gerald Sussman. Guy Steele's philosophical ideas were a civilizing influence on my thought. Tomas Lozano-Perez, Drew McDermott, Johan de Kleer, Scott Fahlman, Richard Brown, Howard Shrobe, Charles Rich, Marilyn Matz, Kurt VanLehn, Richard Waters, Joseph Schatz, and Marvin Minsky helped with many discussions. Gerald Sussman, Guy Steele, Johan de Kleer and Chuck Rich supplied many ideas as the first users of the TMS. Johan de Kleer, Marilyn Matz, Richard Stallman, Charles Rich, Candace Bullwinkle, Richard Brown and Patrick Winston helped me debug this document. The Fannie and John Hertz Foundation supported my research with a graduate fellowship.

On the cover: A picture with an undisclosed and purely personal meaning for the author, who is solely responsible for its presence. I thank Karen Prendergast and Kim Chevalier for much assistance in the preparation of the cover.

This research was conducted at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract number N00014-75-C-0643.

CONTENTS

I. Introduction

A. Overview of the Report

The thesis developed here is that reasoning programs which take care to record the logical justifications for program beliefs can apply several powerful, but simple, domain-independent algorithms to

{1} maintain the consistency of program beliefs,

{2} realize substantial search efficiencies, and

{3} automatically summarize explanations of program beliefs.

These algorithms use the recorded justifications to maintain the consistency and well-founded basis of the set of beliefs. The set of beliefs can be efficiently updated in an incremental manner when hypotheses are retracted and when new information is discovered. The recorded justifications also enable the pinpointing of exactly those assumptions which support any particular belief. The ability to pinpoint the underlying assumptions is the basis for an extremely powerful domain-independent backtracking method. This method, called Dependency-Directed Backtracking, offers vastly improved performance over traditional backtracking algorithms.

These techniques of recording and using justifications also indicate methods for structuring the deductive process so that the justification-derived arguments for certain

types of deductions can be automatically summarized. The levels of detail in a hierarchical problem solver can be separated from each other by this summarization. The separation is accomplished by replacing a set of beliefs at one level by the higher-level beliefs from which they derive. This is important in improving the coherence of explanations, and in further improving the search efficiency of dependency-directed backtracking. Modest extensions of this method are useful in automatically generalizing the results of certain forms of computations.

This report describes techniques for representing, recording, maintaining and using justifications for beliefs. In addition, we present an annotated implementation of a domain-independent program making these functions easily available to programs in a wide range of applications.

The first chapter of the report introduces the function and operation of the Truth Maintenance System (TMS). This is a particular domain-independent program embodying the techniques described in the remainder of this report. An example of problem solving using the TMS is presented to lend substance to the following discussion.

The second chapter develops a new method for representing knowledge about beliefs. This representation, called a non-monotonic dependency system, is closely related to representations used in certain methods of natural deduction. It extends these representations by incorporating the ability to represent non-monotonic dependencies. Non-

monotonic dependencies can be used to express certain forms of hypothetical assumptions, as well as ordinary deductions and conditional proofs. This representation is invaluable in maintaining the consistency of program beliefs in the presence of assumptions and in dependency-directed backtracking. The discussion then presents methods for using this representation in describing several common problem solving structures. These include default assumptions, sets of alternatives, and equivalence class representatives. Finally, a method for imposing a hierarchical structure of summarizations on arguments for beliefs is described.

The third chapter details the mechanisms behind the techniques of the preceding chapter. An important problem in using justifications to determine the set of current beliefs is the occurrence of circular proofs for beliefs. The solution of this problem requires the maintenance of well-founded support for all program beliefs. The chapter discusses this problem and the mechanism of truth maintenance used to incrementally derive well-founded support for beliefs following the addition of new justifications and the retraction of premises. Mechanisms for handling conditional proofs are then described. These involve a mechanization of the deduction theorem of mathematical logic to find the grounds for belief in an implication. Conditional proofs play a major role in the following presentation of the mechanism of dependency-directed backtracking.

The final chapter provides a summary discussion of the key ideas developed in the report: the importance and uses of recorded justifications for beliefs. This summary is

followed by discussions of the relation of this research to other work, and a list of topics for future research.

Three appendices provide information on other topics. The first appendix contains a glossary of the concepts and terms employed in our discussion of truth maintenance systems. The second appendix outlines the structure and mechanisms of a related representational system, the monotonic dependency system. This representation allows a simplification of the algorithms used in truth maintenance over the corresponding algorithms used in the non-monotonic system. However, the monotonic system requires substantial additional complexities on the part of other algorithms. In particular, hypothetical assumptions and dependency-directed backtracking require other mechanisms for their implementation. The third appendix presents an annotated implementation of a version of the TMS program.

Footnotes are indicated in this report by a superscript mnemonic. The notes themselves, indexed by these mnemonics, are located immediately preceding the references.

B. A Functional Description of the Truth Maintenance System

The Truth Maintenance System is a program for recording knowledge about deductions. A reasoning program interacts with the TMS by distinguishing a set of program structures as describing the set of program beliefs. These structures are typically those derived by program operation, and then used to derive further such structures. For instance, the set of assertions and procedures in a PLANNER-like data base may be taken as the set of program beliefs. The TMS associates a TMS-node with each of these structures. These nodes are used for recording several types of information about belief in the corresponding program structure. The most important piece of information is the set of justifications for a node. These justifications describe the reasons for believing the knowledge represented by the program structure associated with the node. Each time the reasoning program determines a new belief or assertion, it informs the TMS of a justification for the node corresponding to the new belief. This justification is in terms of the nodes of the other program structures used in the derivation. The TMS then adds this new justification to the set of justifications attached to the TMS-node of the new belief.

Each time a new justification is provided, the TMS checks to see if any changes in beliefs are indicated by the new justification. If so, the process of truth maintenance is invoked. This involves examining the recorded justifications to redetermine well-founded support for the nodes whose justifications depended on the changed beliefs. The program is then notified of the changes made by the TMS. To allow this, the program can associate

several functions with each of the TMS-nodes associated with its own structures. To signal the occurrence of changes in beliefs, the TMS simply invokes the function associated with the changed TMS-node and the nature of the change that has occurred. The program structure associated with the changed node is used as the argument passed to the function. In this way, the program can arrange that changes in its beliefs can initiate any desired changes to its structures.

The program can also tell the TMS that a certain node represents a contradiction. The TMS remembers this, and attempts to ensure that the node is never believed. It does this by invoking the dependency-directed backtracking system whenever well-founded support is derived for the contradiction node. The backtracker will then try to invalidate the support of the contradiction by removing one of the assumptions underlying the contradiction. Contradictions which cannot be removed by this process are tolerated.

The TMS provides many useful functions for interrogating the structure of the current set of program beliefs. Whenever well-founded support for a TMS-node is determined, the TMS also records a set of antecedent nodes and a set of consequent nodes. These sets are derived from the justifications used in determining the well-founded support. The set of antecedents is the set of nodes on which the belief depends. The set of consequences is the set of nodes whose justifications depend on the belief. The TMS may then be queried for the antecedents or consequences of a particular belief, or for other information derived from these relationships between beliefs. For instance, the set of

premises and the set of assumptions underlying a belief are easily obtained by examining these relationships. Likewise, the entire set of nodes upon which a node depends, or which depend on the node can be easily computed.

C. An Example

On the ground,
Sleep sound.
I'll apply
To your eye,
Gentle lover, remedy.
When thou wakest,
Thou takest
True delight
In the sight
Of thy former lady's eye.
And the country proverb known,
That every man should take his own,
In your waking shall be shown.
Jack shall have Jill,
Nought shall go ill,
The man shall have his mare again and all shall be well.

William Shakespeare, *A Midsummer Night's Dream*

This section presents a simple example involving the making of assumptions, truth maintenance, and dependency-directed backtracking. For this example we set modesty aside and attempt to imitate William Shakespeare in designing the plot of *A Midsummer Night's Dream.* The play is to be a comedy. The major problem in this undertaking is to depict the foolishness of mortals while preventing the story from turning into a tragedy. This is done with a truth maintenance system. The structure of the plot, in terms of the player's attitudes, is determined by truth maintenance. When events threaten tragedy, dependency-directed backtracking is invoked to change their attitudes. These steps eventually determine a consistent (happy) set of attitudes for the players. Disappointingly, this report can only describe the mechanisms behind truth maintenance and dependency-

directed backtracking. We must surrender the explanation of the magic of Puck to future research.

The problem involves the four individuals Demitrius, Helena, Hermia, and Lysander. Initially, Hermia loves Lysander, Helena loves Demitrius, and to make the story interesting, both Demitrius and Lysander love Hermia. We first specify the loves of the women to our reasoning program.

```
(Assert (loves Hermia Lysander) (Premise))
F-1 (LOVES HERMIA LYSANDER) (PREMISE)

(Assert (loves Helena Demitrius) (Premise))
F-2 (LOVES HELENA DEMITRIUS) (PREMISE)
```

The information and rules of our example will be framed in the AMORD[AMORD] problem solving system. Assertions, as above, are of the form (ASSERT <assertion pattern> <justification>) and should be read as "belief in <assertion pattern> is justified by <justification>." The justifications refer to functions which will accept the information transmitted in the justifications and implement the necessary TMS justifications between facts. Facts are referenced in justifications by means of a unique name of the form "F-nn" for each fact.

The next specification is that of the amatory preferences of the men. In contrast to the solid beliefs of the women, the men are easily swayed by flowers and dependency-directed backtracking.

```
(Assume (loves Demitrius Hermia) (Premise))
F-3 (ASSUMED (LOVES DEMITRIUS HERMIA)) (PREMISE)
F-4 (NOT (LOVES DEMITRIUS HERMIA)) ()              ;No justification specified
F-5 (LOVES DEMITRIUS HERMIA) (ASSUMPTION F-3 F-4)
```

Assumptions are the fundamental use of non-monotonic justifications in the dependency system. The assumption of F-5 above is accomplished by asserting the reason (F-3) for the assumption and establishing belief in F-5 based on this reason and on the lack of belief in F-4. This mechanism will be explained in more detail in the next chapter. Its effect is to ensure that F-5 will be believed as long as there are no reasons for believing otherwise. At this point, F-4 is not believed, for no reasons exist supporting its belief. F-5 is believed, since F-3 is believed and F-4 is not.

```
(Assume (loves Lysander Hermia) (Premise))
F-6 (ASSUMED (LOVES LYSANDER HERMIA)) (PREMISE)
F-7 (NOT (LOVES LYSANDER HERMIA)) ()
F-8 (LOVES LYSANDER HERMIA) (ASSUMPTION F-6 F-7)

(Rule (:n (not (loves Demitrius Hermia)))
      (Assert (loves Demitrius Helena) (Quality-not-quantity :n)))
```

This rule specifies Demitrius' love if he falls from love with Hermia by providing the alternative of Helena. The first component of the rule is a pattern, which specifies both a variable (marked by the colon prefix) to be bound to the fact name of the matching assertion, and the pattern against which assertions are to be matched. The body of the rule follows the pattern. If a matching assertion is present, the rule will bind the variables of the pattern to the values derived from the match and evaluate each expression of the body. If it becomes known that Demitrius does not love Hermia, the above rule will justify the belief that Demitrius loves Helena. Lysander's second choice is described similarly.

```
(Rule (:n (not (loves Lysander Hermia)))
      (Assert (loves Lysander Helena) (Love-in-idleness :n)))
```

Next, we add some real-world knowledge about the troubles of men.

```
(Assert (jealous Lysander) (Premise))
F-9 (JEALOUS LYSANDER) (PREMISE)

(Rule (:j (jealous :x))
      (Rule (:l1 (loves :x :y))
            (Rule (:l2 (loves :z :y))
                  (if (not (equal :x :z))
                      (Assert (kills :x :z) (Jealousy :j :l1 :l2))))))
```

This rule embodies the knowledge that jealous people tend to react unpleasantly against others who also love the object of their jealousy. The conditional of the rule body ensures that jealousy is not self-applicable.

```
(Rule (:l1 (loves :x :y))
      (Rule (:l2 (loves :y :z))
            (if (not (equal :x :z))
                (Assert (kills :x :x) (Unrequited-love :l1 :l2)))))
```

This rule expresses the depression and consequent action resulting from unrequited love.

The final rule provides the means by which the happy nature of this comedy is ensured. This is done by declaring all murders to be tragedies. The occurrence of a tragedy is declared to be a contradiction. (Actually, tragedies can be interpreted as signalling the contradiction of a particular murder with the general principle (not (kills :x :y)).) This contradiction will lead to changing the set of assumptions about the loves of the characters which lead to the tragedy.

```
(Rule (:k (kills :x :y))
      (Assert (tragedy :k) (Contradiction :k)))
```

With these assertions and rules we begin the analysis of the conflicts between the desires of the four lovers. AMORD does not specify the order in which rules are to be applied to matching assertions. We will choose an order of application which provides for maximal entertainment.

The first derived assertion notes the conflict caused by Lysander's jealousy.

```
F-10 (KILLS LYSANDER DEMITRIUS) (JEALOUSY F-9 F-8 F-5)
```

This is noticed to be a tragedy, and so ruled out as a happy state of affairs.

```
F-11 (TRAGEDY F-10) (CONTRADICTION F-10)
```

The derivation of belief in a contradiction indicates the inconsistency of the set of beliefs used in deriving the contradiction. To restore the (apparent) consistency of the set of beliefs, the TMS notifies the dependency-directed backtracker of the contradiction. The backtracking process consists of tracing backwards through the antecedents of the contradiction to find the set of assumptions underlying the contradiction. One of these assumptions must be removed to remove the contradiction. The proper justification for the removal must be specified. The reason for retracting an assumption is that the assumption, when combined with the other assumptions, provides support for the contradiction. This reason is valid only under certain circumstances -- those in which the combination of the set of assumptions provides support for the contradiction. This is the statement of a conditional proof. That is, the justification for not believing a particular assumption is that

the other assumptions are believed, and that if all the assumptions are believed, the contradiction follows. Dependency-directed backtracking improves on traditional backtracking mechanisms in two ways; irrelevant assumptions are ignored, since the set of inconsistent beliefs is determined by tracing dependencies; and the cause of the contradiction is summarized in terms of this set of inconsistent assumptions as a conditional proof which remains valid after the contradiction itself has been removed.

In the case at hand, this process begins by examining the reasons for the contradiction in order to locate the inconsistent set of assumptions underlying the contradiction. The contradiction F-11 depends upon F-10, which in turn depends upon F-9, F-8, and F-5. F-8 and F-5 are recognized as assumptions by the system, since the reasons for their beliefs include the lack of belief in F-7 and F-4 respectively. Beliefs supported by a lack of knowledge in other assertions are suspect. This is because an inconsistency based on the lack of a reason for believing some fact can be interpreted as providing a reason for believing that fact. For instance, a robot may decide that it is safe to cross the street because of its failure in attempting to prove that it is unsafe to cross the street. If the robot then gets run over, that contradicts its belief in the safety of its actions. The natural conclusion is that it was unsafe to cross the street.

The backtracking system will try to reconcile the conflicting assumptions by making sure they are not all believed at once. This is accomplished by choosing one of the suspect assumptions at random and disbelieving it. The disbelief is brought about by

justifying belief in one of the supporting facts whose former lack of valid justifications allowed belief in the selected assumption. This new justification is made on the basis of the beliefs in the other assumptions underlying the contradiction and that part of the support for the contradiction which does not depend on these assumptions.

Note at this point one of the efficiencies of dependency-directed backtracking relative to the traditional chronological backtracking schemes. In the above, the set of inconsistent assumptions underlying the contradiction is a subset of all extant assumptions, for I neglected to mention my assumptions about the loves of Theseus, Hippolyta, Oberon, Titania, Bottom, Pyramus and Thisby. These other assumptions may have been determined after the current choices for Lysander and Demitrius. Chronological backtracking systems for choosing alternatives might search through sets of choices involving these independent assumptions. The dependency-directed system will only consider those assumption actually affecting the discovered contradiction.

The next step in the backtracking procedure is the creation of a NOGOOD,[NOGOOD] an assertion summarizing the support for the contradiction which is independent of the inconsistent set of assumptions.

F-12 (NOGOOD F-11) (CP F-11 (F-8 F-5))

This statement of independent support is made by means of a conditional proof justification, stating that F-12 should be believed if when F-8 and F-5 are believed, so is F-11. In the present situation, this reduces to the question of belief in F-9. This is because

F-9 can be combined with the assumptions F-8 and F-5 to support belief in Demitrius' murder. In effect, belief in F-12 is supported solely by belief in Lysander's jealousy.

We must not believe all the assumptions in the set of inconsistent assumptions at the same time. To ensure this, the NOGOOD is used to justify belief in some of the assertions underlying these assumptions. The assertions to justify are those whose lack of belief was used in the support of the assumptions. The minimum that is necessary to accomplish this is to provide a valid justification for belief in one of the unbelieved assertions supporting one of the assumptions. Logically, any one of the assumptions in the inconsistent set can be removed. However, unimpeachable reasons may later be found for believing some of the assumptions. It is necessary to make sure that each assumption will be retracted in turn if the remaining assumptions cannot be doubted. By describing each of the possible ruling-out of beliefs through these new justifications, knowledge of the inconsistency is preserved even if an assumption is retracted and later rejustified.

```
F-7 (NOT (LOVES LYSANDER HERMIA)) (NOGOOD F-12 F-5)
TRUTH MAINTENANCE PROCESSING DUE TO F-7.
  F-4 (NOT (LOVES DEMITRIUS HERMIA)) (NOGOOD F-12 F-8)
```

Note that truth maintenance occurred after the new support for belief in F-7, since belief in F-8 depended on a lack of belief in F-7. The invocation of truth maintenance affected only those beliefs determined from the changed belief -- F-7, F-8, F-10, and F-11. All other facts are known, by means of the recorded dependencies, to be independent of these changes. Following truth maintenance, F-7 is believed, and F-8, F-10, and F-11 are not. Since F-8 is not believed, the following justification of F-4 via F-12 and F-8 fails to support belief in

F-4.

With the backtracking concluded, we can continue the analysis of the consequences of the rules. The next focus for attention is Lysander's change of lover. His love for Hermia was retracted by the backtracker, so he now turns to Helena. Unfortunately, this means that Hermia has now lost her love and kills herself in a fit of despondence.

```
F-13 (LOVES LYSANDER HELENA) (LOVE-IN-IDLENESS F-7)
F-14 (KILLS HERMIA HERMIA) (UNREQUITED-LOVE F-1 F-13)
F-15 (TRAGEDY F-14) (CONTRADICTION F-14)
```

Another bout of backtracking is invoked. This time, tracing backwards through the antecedents of the contradiction leads to F-14, F-13, F-1, F-7, F-12, and F-5. Of these, only F-5 is an assumption. The NOGOOD mechanism then forces the retraction of the assumption that Demitrius loves Hermia.

```
F-16 (NOGOOD F-15) (CP F-15 (F-5))
F-4 (NOT (LOVES DEMITRIUS HERMIA)) (NOGOOD F-16)
   TRUTH MAINTENANCE PROCESSING INVOKED BY F-4.
```

In this situation the support of the NOGOOD consists of the beliefs F-1 and F-12. This invocation of truth maintenance involves checking the beliefs in F-4, F-5, F-7, F-8, F-13, F-14, and F-15. The supporting of belief in F-4 now removes support for F-5, which was the reason for the retraction of Lysander's love for Hermia. Truth maintenance determines that F-4 and F-8, are believed, and that F-5, F-7, F-13, F-14, and F-15 are not. This means that Lysander is now back with Hermia.

The end of the example comes as Demitrius now sees the error of his ways (with a little help from the backtracking system).

F-17 (LOVES DEMITRIUS HELENA) (QUALITY-NOT-QUANTITY F-4)

This satisfies Helena's love. Since Hermia and Lysander were united in the last invocation of truth maintenance, all is now well.

II. Truth Maintenance Systems Applied

A. Introduction

A truth maintenance system provides a representation for describing properties of beliefs and relationships between beliefs. These properties and relationships are interpreted by the TMS as constraints on the current set of beliefs. The duty of the TMS is to record these constraints and to maintain the current set of beliefs in accordance with these constraints. The following chapter describes mechanisms for efficiently achieving this concordance. This chapter describes the basic types of information accepted by the TMS, and their use in describing more complex relationships of importance in problem solving programs.

The TMS accepts two forms of information; justifications for beliefs, and declarations of contradictions. Justifications specify conditions under which a belief is to be held. These conditions involve questions about whether certain other beliefs are held. Contradictions specify conditions under which a belief is not to be held. Beliefs which are declared to be contradictions indicate that an inconsistent set of hypotheses is in force. The inconsistency is resolved by invoking the process of dependency-directed backtracking.

The second section of this chapter describes the representation of reasons for

holding beliefs. This representation can be used to describe non-monotonic deductions and conditional proofs. The third section uses this representation to model several important relationships between beliefs in problem solving systems. These include default assumptions, sets of alternatives, and equivalence class representatives. The final section describes how beliefs may be automatically organized into hierarchical levels of detail.

B. Representing Knowledge About Beliefs

Everything's got a moral, if you can only find it.

Lewis Carroll, *Alice in Wonderland*

A truth maintenance system has two basic components; beliefs, and justifications for beliefs. A node is used to represent a component of program knowledge which may be invested with belief. For instance, each assertion in a PLANNER-like data base might have an attached node to represent the assertion in the TMS. A justification is used to represent a reason for believing the knowledge represented by a node. Justifications for belief in a node are predicates of other nodes. These predicates have an internal structure which is accessible to the truth maintenance system. This allows the TMS to extract several types of dependency relationships between nodes by examining justifications. The two most important types of dependencies are those of the antecedence and consequence relationships between nodes.

A node may have several justifications for belief. The node is believed if at least one of these justifications is valid. A justification is valid if it evaluates *true*. We say that a node which is believed is *in*, and that a node without a valid justification is *out*. The distinction between *in* and *out* is not that of *true* and *false*. The former denote conditions of knowledge about reasons for belief; the latter, belief in a piece of knowledge or its negation.[Logics of Belief] We say that a node which is *in* has a support-status of IN, and that a

node which is *out* has a support-status of OUT. IN and OUT will also be used as predicates of sets of nodes in justifications.

The basic types of justifications for belief are premise justifications, deductive justifications, conditional proof justifications, and assumption justifications. Nodes believed due to premise, deductive, conditional proof, or assumption justifications are called, respectively, premises, deductions, implications, and assumptions.

Premise justifications correspond to the constantly *true* predicate. Premises are therefore always *in*, independent of any other beliefs. Premises are useful in expressing the basic knowledge of a program, and in hypothetical reasoning.

Deductive justifications express that one belief follows from belief in each node of a set of nodes. Deductive justifications are the most common form of justification in normal computations.

Conditional Proofs are justifications for supporting belief in a node on the basis of the derivability of one node from other nodes. A conditional proof justification has two parts; a single node, called the consequent of the conditional proof, and a set of nodes, called the hypotheses of the conditional proof. The support implied by the conditional proof justification is the subset of the support of the consequent which does not derive from the hypotheses. Nodes justified by conditional proofs have the meaning of an implication.

The most important applications of conditional proof justifications are in summarizations. In dependency-directed backtracking, for example, conditional proofs are used to note the reasons for the inconsistency of a set of assumptions. Even after some of the assumptions are retracted, the conditional proof remains valid. This prevents the same mistake occurring in the future.

Assumption justifications support beliefs on the basis of a lack of knowledge. They are justifications which are valid only if specified nodes are *out*. Assumptions represent non-monotonic knowledge. Unlike other types of justifications, assumption justifications can be invalidated by the addition of new justifications for beliefs. The typical use of an assumption justification is in deriving one belief through an inability to prove it false. For instance, one may assume a node *F true* unless proven otherwise by justifying *F* with the predicate (OUT *~F*). If *~F* is a node representing the negation of *F*, this justification will support belief in *F* as long as there are no valid reasons for believing *~F*.[THNOT]

These types of justification can be captured in two forms of predicates. The first of these, the support-list justification, is represented by a predicate of the form

(AND (IN <inlist>) (OUT <outlist>)).

A support-list justification is valid if each node in its *in*list is *in*, and each node in its *out*list is *out*. Premise justifications are support-list justifications in which both the *in*list and *out*list are empty. Deductive justifications are those in which the *out*list is empty.

Assumption justifications have a non-empty *out*list.

Conditional proof justifications cannot be represented by a support-list justification. Conditional proof justifications will be represented by a predicate of the form

(CP <consequent> <inhypotheses> <outhypotheses>).

A justification of this form is valid if the consequent node is *in* whenever each node of the *in*hypotheses is *in* and each node of the *out*hypotheses is *out*. Standard conditional proofs in natural deduction systems specify a single set of hypotheses. Our conditional proofs require that the set of hypotheses be divided into two disjoint subsets, since nodes may be derived both from some nodes being *in* and other nodes being *out*. Some natural deduction systems also allow a set of consequents in a conditional proof. We restrict our conditional proofs to a single consequent for the purpose of efficiency. Note that if multiple consequents were allowed, in the form of a set of *in* consequents and a set of *out* consequents, support-list justifications would be representable as conditional proofs with empty sets of hypotheses.

C. Problem Solving Structures

> Or, if there were a sympathy in choice,
> War, death, or sickness did lay siege to it,
> Making it momentany as a sound,
> Swift as a shadow, short as any dream,
> Brief as the lightning in the collied night,
> That, in a spleen, unfolds both heaven and earth,
> And, ere a man hath power to say, "Behold!"
> The jaws of darkness do devour it up:
> So quick bright things come to confusion.
>
> William Shakespeare, *A Midsummer Night's Dream*

These basic types of justifications can be employed to represent more complex relationships between beliefs. This section is devoted to describing some of these. The relationships presented below describe choice structures. In these, the justifications are arranged to select one alternative from a set of alternatives. In most cases, this choice is backtrackable. That is, if a contradiction is derived which depends on the choice, the backtracking mechanism can cause a new alternative to be chosen from the set of alternatives. In the equivalence class representative selector, the choice is not backtrackable. In some of the backtrackable choice structures, new alternatives will be chosen in a specified order as previous alternatives are ruled out. In others, random choices are made from the set of yet acceptable alternatives. An additional complexity involves the extensibility of the structures. Some of the structures can be augmented at any time by new members of the set of alternatives. Other structures are fixed at creation, and cannot be augmented.

C.1 Default Assumptions

One very common technique used in problem solving systems is to specify a default choice for the value of some quantity. This choice is made with the intent of overriding it if either a good reason is found for using some other value, or if making the default choice leads to an inconsistency. In the case of a binary choice, such a default assumption can be represented by believing a node if the node representing its negation is *out*. The more general case can be represented by the following generalization of the binary case. Let $\{F_1, \ldots, F_n\}$ be the set of the nodes which represent each of the possible values of the choice. Let G be the node which represents the reason for making the default assumption.[NEEDCHOICE] Then F_i may be made the default choice by providing it with the justification

$$\text{(AND (IN } G\text{) (OUT } F_1 \ldots F_{i-1}\, F_{i+1} \ldots F_n\text{))}.$$

If no information about the choice exists, there will be no reasons for believing any of the alternatives except F_i. Thus F_i will be *in* and each of the other alternatives will be *out*. If some other alternative receives a valid justification from other sources, that alternative will become *in*. This will invalidate the support of F_i, and F_i will become *out*. If a contradiction is derived from F_i, the dependency-directed backtracking mechanism will recognize that F_i is an assumption by means of its dependence on the other alternatives being *out*. The backtracker may then justify one of the other alternatives at random, as described in the following chapter, causing F_i to go *out*. In effect, backtracking will cause the removal of the default choice from the set of alternatives, and will set up a new default

assumption structure from the remaining alternatives.

The above structure is not extensible. No new alternatives can be added to the set once the default assumption justification has been made. Such extensibility is necessary when specifying a number as a default due to the large number of possible alternatives. For cases like this the following structure may be used instead. Retaining the above notation, let $\sim F_i$ be a new node which will represent the negation of F_i. We will arrange for F_i to be believed if $\sim F_i$ cannot be proven, and will set up justifications so that if F_j is distinct from F_i, F_j will imply $\sim F_i$. This is done by giving F_i the justification

(AND (IN *G*) (OUT $\sim F_i$)),

and by giving $\sim F_i$ a justification of the form

(AND (IN F_j) (OUT))

for each alternative F_j distinct from F_i. As before, F_i will be assumed if no reasons for using any other alternative exist. Furthermore, new alternatives can be added to the set simply by giving $\sim F_i$ a new justification corresponding to the new alternative. This structure for default assumptions will behave as did the fixed structure in the case of an unselected alternative receiving independent support. Backtracking, however, has a different effect. If a contradiction is derived from the default assumption supported by this structure, $\sim F_i$ will be justified so as to make F_i become *out*. If this happens, no alternative will be selected to take the place of the default assumption. The extensible structure requires an external mechanism to construct a new default assumption whenever the default is ruled out.

C.2 Unordered Sets of Alternatives

Another common problem solving structure is the unordered set of alternatives. Such unordered sets occur in Micro-PLANNER as the set of methods retrieved for solving a goal when no recommendation lists are specified. This structure can be represented as a number of overlayed default assumptions as described in the previous section. By setting up justifications as follows, one of the assumptions will be chosen at random. As before, let $\{F_1, \ldots, F_n\}$ be the set of nodes representing the alternative choices, and let G be the node representing the reason for making the choice. One node of the set of alternatives will be randomly chosen to be *in* if each F_i is provided with the antecedent

(AND (IN G) (OUT $F_1 \ldots F_{i-1} F_{i+1} \ldots F_n$)),

that is,

(AND (IN <reason for alternative set>) (OUT <other alternatives>))

With this structure, the alternative which is selected will be believed unless either a contradiction causes another alternative to be believed, or if one of the other alternatives receives an independent justification for belief. The derivation of a contradiction from the selected alternative will cause another alternative to be selected at random. This in turn will cause the retraction of the previous choice.

This structure does not prevent more than one of the F_i from being *in* via an independent means of support. To impose this exclusiveness, it is necessary to state that each pair of alternatives is inconsistent. This can be done by supporting a contradiction

with a justification (IN F_i F_j) whenever each of the pair of alternatives F_i and F_j becomes *in*.

This structure does not allow the addition of new alternatives to the set of alternatives. To effect such extensibility, the following structure is necessary. For each possible alternative F_i, two new nodes, PA_i (meaning "F_i is a possible alternative") and NSA_i (meaning "F_i is not the selected alternative") should be created. Each PA_i should be justified with the reason for having F_i in the set of alternatives. Each F_i and NSA_i should be justified as follows:

F_i: (AND (IN PA_i) (OUT NSA_i))

{or: (AND <is alternative> <is selected alternative>)}

NSA_i: (IN F_j) {for each j distinct from i}

{or: <no other alternative selected>}

New alternatives can be added to the set by collecting all existing alternatives and creating the above justifications for the the new alternative node and for all of the not-selected-alternative nodes.

C.3 Linearly Ordered Sets of Alternatives

Linearly ordered sets of alternatives are useful whenever heuristic information is available for making a choice. One way such situations arise is by using recommendation lists in Micro-PLANNER. Another use is in heuristically choosing the value of some quantity, such as the state of a transistor or the day of the week for a meeting. These types of sets of alternatives can be described by the following justifications. The justifications are arranged so that backtracking will cause sequencing through the set of alternatives in the specified order. For each alternative A_i, three new nodes should be created. These new nodes are PA_i (meaning "A_i is a possible alternative"), NSA_i (meaning "A_i is not the selected alternative"), and ROA_i (meaning "A_i is a ruled-out alternative"). Each PA_i should be justified with the reason for including A_i in the set of alternatives. Each ROA_i is left unjustified. Each A_i and NSA_i should be given justifications as follows:

A_i: (AND (IN PA_i NSA_1 ... NSA_{i-1}) (OUT ROA_i))

{or: (AND <is alternative> <no better is selected> <is not ruled out>)}

NSA_i: (OUT PA_i) , (IN ROA_i)

{or: (OR <is not a valid alternative> <is ruled out>)}

With this structure, processes can independently rule in or rule out an alternative by justifying the appropriate alternative node or ruled-out-alternative node.

This structure is also extensible. New alternatives may be added simply by constructing the appropriate justifications as above. These additions are restricted to appearing at the end of the order. That is, new alternatives cannot be spliced into the linear order between two previously inserted alternatives.

C.4 Equivalence Class Representatives

In many cases, a value may be computed independently by several different methods. Sometimes different values will be computed for the same quantity. It is desirable to check the consistency of each of these values with all other values for the same quantity. This is a way in which different parts of program knowledge can constrain each other, either by combining two representations of the same value to compute a new quantity, or by declaring the two representations to be contradictory.[Propagation] Each new representation of the value in question should be compared with the previously discovered representations to check for coincidences and contradictions. The new representation should then be added to the list of alternate representations for checking further representations. Since any value in the set of consistent representations is equivalent to all the others, queries made for the value of the quantity should be answered with a single representative. Successive queries should be answered with the same representative. In addition, since all the possible values are equivalent and consistent, backtracking should not attempt to choose a new representive if the selected representative is used in deriving a contradiction.

The following structure describes a mechanism whereby a single representative can be chosen from a set of equivalent objects. The justifications are arranged so that one member of the set will be distinguished as the representative member. This representative will not depend upon the choice from the set. This is done as follows. For each member, M_i, create two new nodes PR_i (meaning "M_i is a possible representative") and SR_i

(meaning "M_i is the selected representative"). Each PR_i should be justified with the reason for believing M_i to be a consistent member of the equivalence class. Each M_i and SR_i should be justified as follows:

SR_i: (AND (IN PR_i) (OUT SR_1 ... SR_{i-1}))

{or: (AND <is a member> <no previous is selected>)}

M_i: (CP of SR_i relative to (OUT SR_1 ... SR_{i-1}))

{or: (CP of <selected alternative> relative to <those not selected>)}

The alternative mechanism selects one of the members PR_i as the representative SR_i. The choice is hidden from the backtracking system by the conditional proof justifications for the representatives. These conditional proofs remove all dependence of the representative on the choice mechanism, so that M_i depends only on PR_i. Because of this, the backtracking system will not attempt to select another member of the set if the selected representative is involved in an inconsistency. The representative will be changed only if an assumption is retracted which supports the membership of the representative in the set. This structure is extensible. New members of the equivalence class can be added to the end of the list of members.

E. Generalization and Levels of Detail

> The lunatic, the lover, and the poet,
> Are of imagination all compact.
>
> William Shakespeare, *A Midsummer Night's Dream*

Hypothetical reasoning is useful in modifying arguments for beliefs. Conditional proofs allow the relativization of beliefs with respect to a set of beliefs. If the original belief is justified by the support of this conditional proof, the argument for the belief has been generalized. Alternatively, the belief may be justified by the conjunction of the support for the conditional proof and a set of beliefs which is semantically equivalent to the set of hypotheses of the conditional proof. If the new set of beliefs is smaller than the set of replaced hypotheses, the argument has been summarized.

One technique for solving a problem is to generalize from the solution of a particular instance of a problem to a solution of the problem itself. This technique is useful when a result valid for each member of a set is desired, but computations can be performed only on specific members of the set. Conditional proofs may be used in such cases to generalize the justification of the specific result by removing its dependence on the specific member used. For example, an electronic circuit analysis program might require a specific (numerical) input voltage to calculate the gain of a circuit. This would produce a specific gain which depends on the particular input voltage used. Actually, the computed gain is

valid over the entire linear region of the device. The gain for all input voltages can be computed by using a typical numeric value of the input voltage to compute an instance of the gain, and then generalizing the argument for this instance. This is done by justifying the general gain value as the conditional proof of the specific gain value relative to the particular value of the input voltage used. (This neglects the problems introduced by the dependence of the computation on inequalities.)

This technique of generalization is a special case of a powerful method for separating levels of detail. This method uses conditional proofs to support results in terms of the names of the methods used to compute them. In this way, the lower level of detail is summarized and replace by the name of the method which produced it. This technique is critically important in hierarchical systems employing truth maintenance. Without such summarizations, explanations of results involve huge numbers of intermediate results from the lower levels of detail used in computing the final result. This not only produces incredibly long and incomprehensible arguments, but also degrades the effectiveness of backtracking and other processes which must trace through arguments for beliefs.

One way to introduce hierarchical structure into computations is to separate knowledge into levels of detail. These levels can be reflected in the dependency structure by a mechanisms analogous to function calling in programming languages. When one level of knowledge needs to use a lower level to compute something, the higher level "calls" a lower level routine. The components of higher level knowledge to be used in the computation are

the "arguments" of the function call. These are mapped into corresponding components of lower level knowledge. These corresponding components represent the "parameters" of the lower level routine. The low level routine, the "function", then performs the desired computation using the parameters, and passes the resulting values back up to the higher level of knowledge.

This analogy is implemented as follows. Routines are attached to named boundaries of knowledge. A boundary consists of two sides and a set of paired terminals. Each terminal on one side of the boundary has a corresponding terminal on the other side. One side of a boundary represents the external view of the routine, that is, the "call" on the routine. The other side of the boundary represents the internal view of the routine, that is, the "parameter declarations" of the routine. When knowledge appears at one side of a terminal of a boundary, it is transmitted to the corresponding terminal on the other side of the boundary. A value is transmitted from an outside terminal by justifying the same value for the corresponding inside terminal. The justification used is in terms of the value's appearance on the outside terminal in conjunction with a node representing the name of the boundary. A value is transmitted from an inside terminal to the corresponding outside terminal by justifying the outside value as the conjunction of the node representing the boundary name and the conditional proof of the inside value with respect to the boundary name. By using the conditional proof, all dependence of the value on knowledge inside the boundary is removed. This knowledge is replaced by adding the boundary name back into the justification after the conditional proof has been performed.[Interfaces]

The following example demonstrates the use of this technique in a hypothetical hierarchical electronic circuit analysis program. This example is loosely based on an actual program (that ran at least once) written by G. J. Sussman, M. Matz, and myself.

The basis of the example will be the conservation of current by a resistor. Resistors are modelled by overlaying two constraints on the basic resistor device.[Slices] These are the constraints of 2-terminalness (conservation of current) and ohms-lawness.

```
(Rule (:t (type :r resistor))
      (Assert (type :r 2-terminal) (Overlay :t))
      (Assert (type :r ohms-law) (Overlay :t)))
```

In this example we will ignore the ohms-law aspect of the resistor. The implementation of the 2-terminal constraint is in terms of a lower level of detail. This lower level implements equations as adders and multipliers connected together. The arithmetic level of devices is used to compute and propagate numerical constraints. This computation is uninteresting to the higher levels dealing with electrical laws and devices. The mechanism of conditional proof will be used to isolate the higher level of detail from the lower.

```
(Rule (:t (type :d 2-terminal))
      (Assert (implementation :d (arithmetic-boxes 2-terminalness))
              (Method :t))
      (Rule (:impl (implementation :d (arithmetic-boxes 2-terminalness)))
            (Assert (type (kcl :d) adder) (part :t :impl))
            (Assert (map (current (#1 :d)) (a1 (kcl :d)) :impl)
                    (Make-map :t :impl))
            (Assert (map (current (#2 :d)) (a2 (kcl :d)) :impl)
                    (Make-map :t :impl))
            (Assert (value (sum (kcl :d)) 0)
                    (Implementation :t :impl))))
```

Here the 2-terminal constraint is implemented as an adder. This adder constrains the

currents of the terminals of the resistor to sum to zero by first mapping each of the currents to an addend of the adder, and then declaring the sum of the adder to be zero. These mappings will serve as the boundary separating the electrical level of detail (the currents on resistor terminals) from the arithmetical level of detail (the connections of adders).

The following rule defines the behavior of the boundary mappings.

```
(Rule (:m (map :var1 :var2 :impl))
      (Rule (:f (value :var1 :val))
            (Assert (value :var2 :val) (INMAP :f :m)))
      (Rule (:f (value :var2 :val))
            (Assert (value :var1 :val) (OUTMAP :f :impl))))
```

This rule sets up the channels by which pieces of information, in this case value assertions, are transmitted across the boundary. The INMAP justification does nothing special; it merely transmits the value across, adding the map assertion to the justification in passage. The OUTMAP justification is the means by which the higher level is separated from the lower level. The OUTMAP justification operates by asserting the value being transmitted at the higher level. The justification removes the dependence on the lower level of detail by using the conditional proof of the lower level value node relative to the implementation node. That is

```
(OUTMAP :f :impl)
```

translates into

```
(CP :f (:impl) ()).
```

Since both the mapping structure and the lower level internal structure depend upon this implementation node, they are absent from the resulting explanations.

As a particular example, we demonstrate the passage of current through a resistor.

```
(Assert (type r1 resistor) (Premise))
F-1 (TYPE R1 RESISTOR) (PREMISE)
F-2 (TYPE R1 2-TERMINAL) (OVERLAY F-1)
F-3 (TYPE R1 OHMS-LAW) (OVERLAY F-1)
```

This defines a particular resistor, R1. The next level of detail is then implemented.

```
F-4 (IMPLEMENTATION R1 (ARITHMETIC-BOXES 2-TERMINALNESS)) (METHOD F-2)
F-5 (TYPE (KCL R1) ADDER) (PART F-2 F-4)
F-6 (MAP (CURRENT (#1 R1)) (A1 (KCL R1)) F-4) (MAKE-MAP F-2 F-4)
F-7 (MAP (CURRENT (#2 R1)) (A2 (KCL R1)) F-4) (MAKE-MAP F-2 F-4)
F-8 (VALUE (SUM (KCL R1)) 0) (IMPLEMENTATION F-2 F-4)
```

With the wiring of the resistor completed (the wiring of the ohms-law constraint has been omitted for brevity), we can specify the current on one side of the resistor and examine the resulting explanations.

```
(Assert (value (current (#1 r1)) 7) (Premise))
F-9 (VALUE (CURRENT (#1 R1)) 7) (PREMISE)
F-10 (VALUE (A1 (KCL R1)) 7) (INMAP F-9 F-6)
F-11 (VALUE (A2 (KCL R1)) -7) (SUBTRACTION F-5 F-10 F-8)
F-12 (VALUE (CURRENT (#2 R1)) -7) (OUTMAP F-11 F-4)
```

The lower level rules implementing the arithmetic constraints of addition and multiplication have also been omitted for brevity.

A query for the explanation of the last node, F-12, produces the following result, in which the level of arithmetic detail is absent by means of the OUTMAP conditional proof of the computed value, as given in F-11, relative to the implementation node F-4. The conditional proof is used to derive the set of higher level beliefs supporting the computed

value. These are the nodes used in the explanation. (The mechanism for deriving sets of supporting beliefs from conditional proofs is described in Chapter III.) Since there is only one possible implementation of a resistor, the IMPLEMENTATION node does not appear in the explanation. In other devices, such as transistors, there may be several possible implementations depending on the state of the device. In such cases, the implementation choice should be included in the justification for the higher-level result.

```
(Explain 'F-12)

PROOF OF F-12 = (VALUE (CURRENT (#2 R1)) -7) (OUTMAP F-11 F-4)

;;;  THE INDEPENDENT SUPPORT OF (OUTMAP F-11 F-4) IS (IN F-9 F-2 F-1).

F-9 (VALUE (CURRENT (#1 R1)) 7) (PREMISE)
F-2 (TYPE R1 2-TERMINAL) (OVERLAY F-1)
F-1 (TYPE R1 RESISTOR) (PREMISE)
```

This example has indicated the usefulness of conditional proofs in clarifying explanations by separating levels of detail. More important benefits are possible in improving the information examined by backtracking systems. Just as concise explanations are more useful to humans, improved explanation structures relating beliefs can ease the task of dependency-directed backtracking. Concise explanations reduce the number of nodes involved in the support of a contradiction. This allows the backtracking system to operate more efficiently. This also means that better reasons are developed in summarizing the reasons for inconsistencies.

III. Truth Maintenance Mechanisms

A. Introduction

The previous chapter presented the descriptive language for imparting information about beliefs to a truth maintenance system. Descriptions in this language concern logical, time-independent relationships between beliefs. Particular descriptions can be given to the TMS in any order. The purpose of this chapter is to describe mechanisms by which the TMS can maintain the consistency of the set of beliefs with the constraints imposed by new justifications and declared contradictions.[Complexity]

This section of this chapter describes the basic components of the world of the TMS; nodes and justifications for belief. The second section presents the problems caused by circularities in dependency relationships. Circularities arise through circular proofs, and require careful handling in the process of determining the support of a belief. The third section discusses the process of truth maintenance. This process is invoked whenever new justifications or the retraction of premises cause beliefs to change. These changes mean that the support of any beliefs affected by the changes must be rederived. The fourth section discusses the handling of conditional proofs. The fifth section combines all these mechanisms to describe dependency-directed backtracking.

Little elaboration is needed on the concepts of nodes and justifications introduced in Section II.B. The internal structure of justifications is accessible by the TMS to allow efficient processing. To the TMS, a support-list justification (hereafter referred to as a SL-justification) is simply a pair of lists of nodes, the *in*list and the *out*list. A conditional proof justification (a CP-justification) has a single consequent node, and two lists of nodes, the *in*hypotheses and the *out*hypotheses. The set of justifications of a node is called its justification-set. From these justifications we derive a dependency relationship between nodes. Each node has a list of consequences. These consequences are other nodes which have justifications mentioning the antecedent node. Each consequent node has either a SL-justification containing the node in question in the *in*list or the *out*list, or has a CP-justification containing the node as either the consequent of the conditional proof, or as one of the hypotheses.

B. Well-Founded Support Relations

Clear your mind of cant.

Samuel Johnson

Consider the situation in which the node *F* represents the assertion

"(= (+ X Y) 4)",

G represents

"(= X 1)",

and *H* represents

"(= Y 3)".

If both *F* and *G* are *in*, then belief in *H* can be justified by (AND (IN *F G*) (OUT)). This justification will cause *H* to become *in*. If *G* subsequently becomes *out* due to changing hypotheses, and if *H* becomes *in* by some other justification, then *G* can be justified by (AND (IN *F H*) (OUT)). Suppose the justification supporting belief in *H* then becomes invalid. If the decision to believe a node is based on a simple evaluation of each of the justifications of the node, then both *G* and *H* will be left *in*. This happens because the two justifications form circular proofs for *G* and *H* in terms of each other. These justifications are mutually satisfactory if *F*, *G* and *H* are *in*.

Belief in nodes on the basis of circular proofs can be avoided by only believing nodes for which there exists a non-circular argument from premises and assumptions. This is accomplished by distinguishing a supporting-justification in the justification-set of each *in* node. The supporting-justification is a valid justification whose validity was determined by examining only nodes with well-founded support. Once a node is believed on the basis of a supporting-justification, it continues to be believed until its supporting-justification becomes invalid. If the node's supporting-justification becomes invalid, the node and any nodes using the node in their well-founded support are examined to see if other justifications can provide new means of support.

It is possible to generalize the supporting-justification of a node to a set of justifications, each of which provides well-founded support for belief in the node. This scheme is attractive because it allows one form of work to be avoided. A node will be believed as long as there is at least one supporting-justification. This makes it unnecessary to reconsider the support of the consequences of a node if the invalidation of one of the node's supporting-justifications leaves the set of supporting-justifications nonempty. There are several problems with this approach. One problem is that the support of node must be checked with each change in its set of supporting-justifications, even if there is one supporting-justification which is always valid. If only one supporting-justification was maintained, this invariant justification could allow the checking process to completely ignore the node. Another problem is that a major use of the recorded justifications is in tracing backwards through the arguments for belief in a node. If multiple supporting-justifications

are used, then processes like dependency-directed backtracking and explanation generation can become very complex and costly. To handle the multiple justifications for beliefs, these processes would have to trace many branching arguments in parallel. Because of these problems, the approach used here is that of maintaining a single supporting-justification for each believed node.

If any of several justifications can provide well-founded support, the best choice for the supporting-justification is the justification which will remain valid the longest. The problem of determining which justification is the most stable is intractable in general, like the problem of selecting pages to swap out of memory. Rather than select a supporting-justification completely at random, some (rather dubious) heuristics can be used. One simple heuristic, the one used in the programs of Appendix 3, is to choose the chronologically oldest of the possible justifications. This heuristic is based on the theory that chronologically older justifications are likely to be more "fundamental" in some sense, and thus less susceptible to change. Alternatively, a "self-organizing" heuristic such as bubbling the more stable justifications to the front of the order may be used to modify the order of the justification-set of the node. The current research has not included any experimentation to see if benefits accrue from the use of these heuristics, or to see which provides the better performance. One might also imagine schemes in which a "certainty factor" is associated with each node. In such a scheme, the "certainty" of a justification might be computed from the certainty factors of the nodes mentioned by the justification. The heuristic then chooses the justification with the largest certainty factor as the

supporting-justification. Most such measures of certainty require the use of domain-specific knowledge about the meaning of the nodes and justifications involved. In addition, the semantics of a measure which is not a formal probability measure is very unclear. The use of certainty factors has not been persued here for these reasons.

For the purpose of tracing through justifications, it is convenient to extract another dependency relationship from the supporting-justifications of nodes. The antecedents of a node are those nodes which currently support belief in the node. Thus an *out* node has no antecedents, and the antecedents of an *in* node are just those in the (necessarily disjoint) union of the *in*list and *out*list of the node's supporting-justification. (As will be explained in more detail later, CP-justifications never are the supporting-justification of nodes. Instead, a CP-justification is used to generate a new SL-justification summarizing the independent support of the conditional proof. This SL-justification is then used as the supporting-justification.)

The antecedence dependency is not the only useful dependency relationship of this nature. When beliefs change, it is necessary to check not only the believed nodes which depended on the changed beliefs, but also any nodes which might now be believed by virtue of the changes. To make this efficient, the set of nodes affecting the current support-status of each node is collected into the node's set of supporting-nodes. The supporting-nodes are the same as the antecedents for *in* nodes, since any change in one of the antecedents of a node may cause the node to become *out*. The supporting-nodes of an *out*

node are found by selecting one node from each justification of the node. Each of these nodes is selected because its support-status led to the invalidity of the corresponding justification. This means that each of the *out* supporting-nodes of an *out* node is in the *in*list of one of the node's justifications, and each *in* supporting-node is in the *out*list of some justification. This definition does not specify a unique set of nodes as the supporting-nodes of an *out* node, but any such set suffices. The intent is for the supporting-nodes of a node to be a small set of nodes which, if unperturbed, indicates that belief in the supported node cannot change. In odd cases this definition allows inclusion into the supporting-nodes of a node some nodes that do not influence the support-status of the node. For example, the silly justification (AND (IN *f*) (OUT *f*)) might cause *f* to be included in the supporting-nodes, even though the justification is constantly false independent of the support-status of *f*. Note also that circularities can occur in the relationship of one node being a supporting-node of another. The algebra example above demonstrates such a case, for if the two nodes involved in the circularity are both *out*, they will have each other as supporting-nodes.

From the supporting-nodes of a node, we define additional concepts as follows. The affected-consequences of a node are those nodes whose current support-status rests on the node; precisely, the affected-consequences of a node are those consequences of the node such that the node is a supporting-node of each of these consequences. The foundations of a node are those nodes involved in the well-founded support for belief in the node; precisely, the foundations of a node are the transitive closure of the antecedents of the node under the operation of taking antecedents. The ancestors of a node are those nodes

involved at some level in determining the current support-status of the node; precisely, the ancestors of a node are the transitive closure of the set of supporting-nodes of the node under the operation of taking supporting-nodes. The repercussions of a node are those other nodes whose support-statuses are affected at some level by the support-status of the node; precisely, the repercussions of a node are the transitive closure of the affected-consequences of the node under the operation of taking affected-consequences.

C. Truth Maintenance

And we'll talk of them too,
Who loses, who wins, who's in, who's out,
And take upon's the mystery of things.

William Shakespeare, *King Lear*

Truth maintenance is a process invoked whenever the support-status of a node changes. This process consists of redetermining the support-statuses of the node and its repercussions. This involves examining the affected nodes to find well-founded support. The presence of several types of circularities in the dependency structure complicates matters. These circularities call for a more elaborate mechanism than a simple bottom-up support analysis.

Nodes can change their support-status in two ways. The normal reason for change is the addition of a new valid justification to an *out* node. This causes a change of support-status from *out* to *in*. The other way that a change can occur is if the justification of a premise is retracted.

Different actions are required depending upon the validity of the new justification and the support-status of the justified node. A justification added to an *in* node requires only adding the new justification to the justification-set of the node. In this case, the new justification cannot cause a change of support or support-status. A new justification added

to the justification-set of an *out* node requires truth maintenance processing if the new justification is valid. A non-valid justification can be added to the justification-set of an *out* node without causing truth maintenance, but this requires that the set supporting-nodes of the node be updated to include a node responsible for the invalidity of the new justification.

Truth maintenance processing is not required if a node has no affected-consequences. The support-status of the node can be changed without affecting any other beliefs. This special case routinely occurs after newly created nodes are given their first justification. Truth maintenance is also not required when retracting a node which does not have a premise justification for its supporting-justification. In this case, the retraction can remove any premise justifications from the justification-set without affecting the node's well-founded support.

Truth maintenance processing starts by producing a list containing the invoking nodes and their repercussions. Each node on this list is marked with a support-status of NIL to indicate that it lacks well-founded support. (A NIL support-status exists only during the process of truth maintenance. Outside of truth maintenance, all nodes have a support-status of either IN or OUT.) This marking is used to determine if a justification supplies well-founded support for a node. Next, each of the nodes on this list must be examined. This is a recursive procedure taking action only if the node being examined has a support-status of NIL. If so, the justification-set of the node is evaluated with respect to well-foundedness. This is a careful evaluation procedure described below. Examination of this

node terminates if well-founded support cannot be found. If well-founded support is found, the support-status of the node is set appropriately, the supporting-nodes are installed, and if the node is now *in*, the supporting-justification is installed. Since the newly determined support of this node might now allow determination of support for its consequences, each of the consequences of the node is examined in turn.[Efficiency]

Justifications can be evaluated in a way which indicates whether they provide well-founded support. This is done by evaluating the justifications with respect to the three values T, F and NIL. SL-justifications evaluate to T if each node of the *in*list is *in* and each node of the *out*list is *out*; to F if some node of the *in*list is *out* or some node of the *out*list is *in*; and to NIL otherwise. CP-justifications evaluate to T if all *in*hypotheses are *in*, all *out*hypotheses are *out*, and the consequent is *in*; to F if the first two conditions hold and the consequent is *out*; and to NIL otherwise. A node is considered *in* if any of its justifications evaluates to T, *out* if all its justifications evaluate to F, and otherwise lacks well-founded support.

The above process of examination will determine well-founded support for the majority of nodes, but can leave some nodes without well-founded support. These are nodes which are involved in circularities in the dependency relation, or whose possible support depends on nodes involved in circularities. There are essentially three different kinds of circularities which can arise. The first and most common is a circularity in which all nodes involved can be considered *out* consistently with their justifications. Such circularities arise

routinely through equivalences and simultaneous constraints. One of these circularities is present in the algebra example of the previous section. In that example, an equation produces a circularity between the nodes *G* and *H*. If neither of *G* or *H* is supported by justifications not involved in the circularity, both *G* and *H* should be considered *out*.

The second type of circularity is one in which at least one of the nodes involved must be *in*. An example is that of two nodes *F* and *G*, such that *F* has an justification of the form (OUT *G*), and *G* has an justification of the form (OUT *F*). Here either *F* must be *in* and *G out*, or *G* must be *in* and *F out*. This type of circularity arises in defining unordered sets of alternatives. The other types of ordered alternative structures avoid such circularities.

The third form of circularity which can arise is the unsatisfiable circularity. In this type of circularity, no assignment of support-statuses to nodes is consistent with their justifications. An example of such a circularity is a node *F* with the justification (OUT *F*). This justification implies that *F* is *in* if and only if *F* is *out*. Unsatisfiable circularities are bugs, indicating a misorganization of the knowledge of the program using the truth maintenance system. Unsatisfiable circularities are violations of the semantics of *in* and *out*, which can be interpreted as meaning that the lack of reasons for belief in a node is equivalent to the existence of reasons for belief in the node. (It has been my experience that such circularities are most commonly caused by confusing the concepts of *in* and *out* with those of *true* and *false*. For instance, the above example could be produced by this

misinterpretation as an attempt to assume belief in the node *F* by giving it the justification (OUT *F*).)

The second step of truth maintenance handles these circularities. This step consists of a relaxation process in which the nodes not supported during the search for well-founded support are specially examined. Like the regular examination procedure, this special examination procedure also ignores nodes possessing a non-NIL support-status. It first checks for well-founded support, as in the previous examination process, and if it finds such support installs it and then processes the node's consequences as described below. If well-founded support is still lacking, the SL-justification set is specially evaluated. This is done by considering a support-status of NIL in a referenced node to be equivalent to OUT. That is, the SL-justification set is evaluated under the assumption that all unsupported nodes are OUT. (The CP-justification set is ignored during this evaluation. This will be discussed later.) This evaluation determines the node to be either *in* or *out*.

Once the examination finds the support-status for the node, it must check the consequences of the node. If the node was determined to be *out*, then a simple recursive examination of the consequences is sufficient. More care is required if the node was brought *in*. Any affected-consequences of the newly supported node had had their support determined on the assumption that the newly supported node was *out*. The *in*ning of this node means that this assumption was mistaken. This requires that the affected-consequences of the node must be remarked and reexamined. If the node had no affected-

consequences, the examination can continue by recursively examining the consequences of the node.

This relaxation procedure will devolve into an infinite loop if unsatisfiable circularities are present. Such circularities, as previously mentioned, are really erroneous uses of the truth maintenance system. This possibility of an infinite loop can be avoided at some expense by making a well-foundedness check before a node is brought *in*. This check operates by checking the ancestors of the *in*ned node to see if they include the node itself. If this condition holds, an unsatisfiable dependency structure has been detected.[Analysis]

The above process is incomplete in its treatment of nodes justified via conditional proofs. CP-justifications are never used as supporting-justifications. They are instead used to generate new SL-justifications. This is done whenever a node is brought *in* on the basis of a CP-justification. The FINDINDEP procedure described in the next section is used to trace backwards from the consequent of the CP-justification to collect the sets of nodes which support the consequent but are not themselves supported by the hypotheses of the CP-justification. These sets (one of *in* nodes, the other of *out* nodes) are then made into a new SL-justification. This new justification is made the supporting-antecedent of the node and added to the justification-set of the node.

CP-justifications are not evaluated if they are out of context, that is, if an *in*hypothesis is *out*, or if an *out*hypothesis is *in*. Rather than perform the hypothesizing of

beliefs necessary to evaluate the conditional proof justification, the justification is ignored. This is an incompleteness in the current system, and a problem for exploration and solution by future research.

The current partial solution to this problem is to pass over the examined nodes after truth maintenance has decided their support-statuses. Each node can have a CP-consequent-list associated with it. This list is used to record the set of other nodes which use the node as a consequent of a CP-justification. If a node has some nodes on its CP-consequent-list and is *in*, then new SL-justifications are derived (if possible) for the nodes possessing the CP-justifications. Those nodes are then justified with any new justifications that result. If this step causes truth maintenance, the scan must be restarted to check for further changed nodes.

A related check performed during this scan is that of looking for *in* nodes marked as contradictions. If such nodes are found, the backtracking mechanisms is invoked and the scan restarted.

Once well-founded support has been derived for all the nodes affected by truth maintenance, the external system can be notified of any changes in beliefs that have occurred. This is easily done by allowing the external system to associate two functions with each node. One of these, the signal-recalling function, is called with the external representation of the node as its argument if the node's support-status has changed from *out*

to *in*. The complementary [illegible] is called if the node's support-status has changed from *in* to *out*.

D. Conditional Proofs

Some mistakes we must carry with us.

Larry Niven, *Ringworld*

The Deduction Theorem of mathematical logic states that if *C* is derivable from *A* and *B*, then (IMPLIES *B C*) is derivable from *A* alone. This theorem forms the basis for the conditional proof mechanism used in a truth maintenance system. This mechanism, called FINDINDEP,[FINDINDEP] is a procedure which uses a proof of the belief *C* to find a justification for (IMPLIES *B C*) in terms of *A*. Conditional proof justifications have a set of *in*hypotheses and a set of *out*hypotheses. If each node in the *in*hypotheses is *in*, and each node in the *out*hypotheses is *out*, then FINDINDEP can be applied to compute the set of support of the implication of any *in* node *C* by these hypotheses. That is, it computes the support of the implication

(IMPLIES (AND (IN <inhypotheses>) (OUT <outhypotheses>)) C).

This is done by finding the set of nodes in the foundations of *C*. This set is pruned by removing all of the *in*hypotheses and *out*hypotheses, as well any nodes that are repercussions of *in*hypotheses or *out*hypotheses. The remaining set of nodes is the set of support for the above implication. This set can be pruned further. Each of the nodes in this set has affected-consequences. Some of the affected-consequences are among the foundations of *C*. If all such affected-consequences of a node are in the derived set of support, the node can be removed from the set of support. This is admissible because the

node supports *C* only through other nodes in the set of support. This pruning leaves a reduced set of support which can be combined with the hypotheses to support *C*.

FINDINDEP consists of two basic steps, each of which is a recursive scan of the foundations of the consequent *C* of the conditional proof. The first step consists of tracing backwards through the antecedents of *C* until *in*hypotheses, *out*hypotheses, or premises are reached. The search then follows the paths traced out in the opposite direction. Tracing upwards from the terminal nodes, the scan marks each of the nodes encountered that is either one of the hypotheses or is a repercussion of a hypothesis. At the end of this step, all unmarked nodes are in the set of support of *C*. The second step finds the reduced set of support. This step again traces backwards through the antecedents of *C*. This tracing stops when an unmarked node is reached, or when a hypothesis is reached. When an unmarked node is encountered, it is included in the reduced set of support. Finally, all of the marks are removed. The nodes collected as the reduced set of support are separated into a set of *in* nodes and a set of *out* nodes. These can be used to create a SL-justification to support the implication.

E. Backtracking

"I should have more faith," he said; "I ought to know by this time that when a fact appears opposed to a long train of deductions it invariably proves to be capable of bearing some other interpretation."

Sir Arthur Conan Doyle, *A Study in Scarlet*

Systems engaging in hypothetical reasoning require mechanisms for reconciling beliefs upon the introduction of new hypotheses. Two types of hypotheses can be distinguished; speculative hypotheses and counterfactual hypotheses. Speculative hypotheses are those which are consistent with existing beliefs and justifications. Speculative hypotheses are useful when a lack of knowledge forces the making of an assumption for the purpose of exploration. Counterfactual hypotheses, on the other hand, contradict previous beliefs. Such hypotheses are useful in exploring the results of actions and in deriving constraints existing in different worlds.

There is an overlap between these two forms of hypotheticals in a truth maintenance system. This overlap results from the orientation of the truth maintenance system towards apparent consistency. Since the only notion of inconsistency is that of contradictions, any set of beliefs not supporting a contradiction node is considered consistent. New information added by the external system may later show these beliefs to be inconsistent. Thus what may have originally been speculative hypotheses may later be discovered to be counterfactual hypotheses requiring special treatment. This treatment is

called backtracking.

The procedure for dealing with contradictions is to determine the set of hypotheses underlying the contradiction. This set is inconsistent, and the inconsistency is resolved by rejecting belief in one or more of these hypotheses. It is desirable to discard as few hypotheses as possible, so the handling of counterfactuals has been characterized as the selection of a maximal consistent subset of the set of inconsistent hypotheses. This process of selection requires domain-specific knowledge, since as far as logic and truth maintenance systems are concerned, premises are independent of all other beliefs. This independence means that there are no inherent relations to other beliefs which can be used in discriminating among premises in consistent subset selection.[PMMCS]

Unlike premises, assumptions <u>can</u> be related to each other and to other beliefs. Assumptions can be related to the reasons for their introduction and to the specific lack of information which permits their belief. An inconsistency involving assumptions indicates not only that one of the assumptions must be retracted, but also that belief is justified in one of the nodes whose lack of valid justifications lead, through the assumption, to the inconsistency. Thus, an inconsistency can be used to derive new information which controls the introduction and consideration of further assumptions.

These mechanisms are embedded in truth maintenance systems in two ways. The making of speculative hypotheticals and the necessary reconciliation of these hypotheses

with previous justifications for belief are handled by the normal mechanisms of assumption justifications and truth maintenance processing. Backtracking is implemented in a truth maintenance system as the method of dependency-directed backtracking. This method uses the dependency relationships to provide the raw material for the analysis and summarization of the inconsistent set of hypotheses.

The first step in the process of dependency-directed backtracking is the recognition of an inconsistency by means of a contradiction. This is a node, justified by the inconsistent beliefs, which the external system declares to represent a false belief. All contradictions have the semantics of *false*, so there need be only one such contradiction node, with new inconsistencies recorded as new justifications for this node. As far as the truth maintenance system is concerned there can be several representations for *false*, each of which is represented by a distinct contradiction node.

The second step of backtracking is the determination of the inconsistent set of hypotheses underlying the contradiction. The wisdom of premises and monotonic justifications is inscrutable to the truth maintenance system. Therefore, the only hypotheses of interest to the backtracker are those based on incomplete knowledge. These are the assumptions. The assumptions are located by tracing backwards from the contradiction node through its antecedents, and watching for nodes which have *out* nodes among their antecedents.

Although the set of all the assumptions supporting the contradiction is easily calculable, recording the inconsistency of this set directly may be inefficient. There is a definite structure relating these assumptions. Since the support of nodes is well-founded, the nodes may be arranged into a partial order. In this partial order, one node is "less than" another node if the first is an antecedent of the second. The complete partial order is derived as the transitive closure of this antecedence-based relation. Some assumptions will be independent of other assumptions in this order. Other assumptions will be dependent on lower level assumptions. This information is useful, because only those assumptions which are maximal in this partial order should be considered for retraction.

An assumption should not be retracted if the contradiction depends on it by means of other assumptions. The backtracking procedure only makes logically necessary retractions based on the observed occurrence of contradictions. In general, there is not enough information to logically rule out lower level assumptions. That is, assumptions are retracted using their inconsistency with the set of other assumptions in force as a reason. The retraction of non-maximal assumptions would then depend on the continued belief in the maximal assumptions which depend on the lower level assumption. Since these higher level assumptions will in general depend crucially on the lower assumption, the attempt at retraction must fail.

Other types of information can also be derived from the partial order. These allow discriminations on the basis of the height of an assumption in the partial order, or on

the size of the component of the partial order containing the assumption. Different strategies for ranking the assumptions correspond to different local search strategies. For instance, retracting the maximal assumption of greatest height in the partial order might be interpreted as a kind of depth-first strategy. However, from a global perspective, dependency-directed backtracking is neither depth-first nor breadth-first, since the global search order is determined primarily by the history of the search, and not by the local order of choosing among alternative assumptions for retraction.

The third step of backtracking is the summarization of the inconsistency of the set of hypotheses underlying the contradiction. Suppose that S = {A, B, ... , Z} is the set of inconsistent assumptions. The backtracker then creates a NOGOOD, a new node signifying that S is inconsistent. The NOGOOD represents the fact that

$$A \wedge \ldots \wedge Z \supset \mathit{false},$$

or alternatively, that

$$\sim (A \wedge \ldots \wedge Z).$$

S is called the NOGOOD-set of the NOGOOD. The summarization is accomplished by justifying the NOGOOD with a conditional proof of the contradiction relative to the set of assumptions. In this way, the inconsistency of the set of assumptions is recorded as a node which will be believed even after the contradiction has been disposed of by the retraction of some hypothesis. Note also that the NOGOOD will depend on any non-maximal assumptions not included in the NOGOOD-set. This means that future backtracking can reject each of the assumptions in the NOGOOD-set and still have some assumptions left to

reject.

The last step of backtracking uses the summarized cause of the contradiction, represented by the NOGOOD, to both retract one of the inconsistent assumptions and to prevent future contradictions for the same reasons. This is accomplished by deriving new justifications for the *out* nodes underlying the inconsistent assumptions. The new justifications will cause one of these *out* facts to become *in*, thereby causing one of the offensive assumptions to become *out*. This step is reminiscent of the justification of results on the basis of the occurrence of contradictions in reasoning by reductio ad absurdum.[Combinatorics]

These new justifications are constructed as follows. Let the inconsistent assumptions be $A_1, \ldots, A_n$. Let $S_{i1}, \ldots, S_{ik}$ be the *out* nodes of the justification supporting belief in the assumption A_i. To effect the retraction of one of the assumptions, A_i, justify S_{i1} with the predicate

(AND (IN *NG* $A_1 \ldots A_{i-1} A_{i+1} \ldots A_n$) (OUT $S_{i2} \ldots S_{ik}$)),

that is,

```
(AND (IN <NOGOOD> <other assumptions involved>)
     (OUT <other denials of this assumption>))
```

This will ensure that the justification supporting A_i by means of this set of *out* nodes will no longer be valid whenever the NOGOOD (*NG*) and the other assumptions are believed. This process may be repeated for each assumption in the inconsistent set to try to ensure

that the contradiction will be removed even if some of the assumptions in the nogood-set have alternate means of support. However, this strategy will create a circularity containing these new justifications. While later backtracking may make this unavoidable, the immediate creation of a circularity can be avoided by making only one new justification. This new justification will neutralize the justification of one of the assumptions. If other support can be found for this assumption, then backtracking is repeated. Presumably the new invocation of the backtracker will find that the previous culprit is no longer an assumption. Backtracking halts when the contradiction becomes *out*, or when no assumptions can be found underlying the contradiction.

IV. Discussion

A. Summary of the Key Ideas

The major point stressed here is that careful recording of the logical support for program beliefs permits many important efficiencies and capabilities in reasoning programs. We feel that truth maintenance systems should be thought of as a systemic function of problem solving systems. Just as pattern-directed data bases form a naturally used subsystem of many problem solvers, so, we feel, should a truth maintenance system.

This report has elaborated the structure and use of a non-monotonic dependency system for representing knowledge about beliefs, the mechanisms by which a truth maintenance system can employ this representation of knowledge to maintain beliefs consistent with recorded justifications, the application of dependency relationships in effecting backtracking, and mechanisms for separating levels of detail and their dependencies.

The non-monotonic dependency system formalizes several ways of justifying beliefs including: premises, beliefs which are independent of other beliefs; deductions, beliefs derived from other beliefs; conditional proofs, beliefs summarizing the derivability of one belief from others; and assumptions, the non-monotonic justifications in which a

belief is based on a lack of contradictory knowledge. These basic representational techniques combine to allow perspicuous implementations of several common belief structures used in problem solving systems, including default assumptions, sets of alternatives and selectors of equivalence class representatives.

Beliefs consistent with recorded justifications can be efficiently determined via truth maintenance, a process invoked whenever beliefs change due to the addition of new information or the retraction of hypotheses. Truth maintenance involves an examination of those beliefs explicitly linked, by means of the dependency system, to the changed beliefs. The truth maintenance system exercises the care required to avoid spurious beliefs produced by circularities among the justifications for beliefs.

Exploiting all the facilities provided by the dependency and truth maintenance systems, dependency-directed backtracking examines the well-founded support recorded for beliefs involved in inconsistencies to determine the set of hypotheses underlying the inconsistency. Retraction of premises supporting an inconsistency is outside the domain of a truth maintenance system, but the dependency relationships involving non-monotonic assumptions can be analyzed to provide a basis for the retraction of assumptions. The causes of the inconsistency can be summarized via a conditional proof, and this summarized cause can then be used to retract one of the underlying assumptions. This is done by providing new knowledge which rules out belief in one of the assumptions.

Finally, the mechanism of conditional proof can be used to separate hierarchical levels of detail in explanations. The hierarchy is maintained by modifying arguments for beliefs. The modification consists of replacing a set of beliefs at one level by the set of higher-level beliefs from which they were derived. This is done by justifying information at one level in terms of the higher-level beliefs and the conditional proof of the corresponding information at the lower levels relative to the higher-level structures. This separation is important not only in improving the clarity of explanations, but in aiding processes like dependency-directed backtracking which must interrogate these explanations. In the case of dependency-directed backtracking, the separation of levels of detail reduces the number of belief involved in an inconsistency, thereby increasing the efficiency of the backtracking process.

B. Relation to Other Work

There are systems oriented towards hierarchical representations of knowledge, but none of these systems deal with recorded justifications for beliefs. None of these systems address the problems raised by integrating the methods of truth maintenance and hierarchical representations of knowledge.

There are several systems employing some form of data dependencies. One class of these uses explicit justifications for belief. These include the systems of Fikes [1975], McDermott [1975, 1977], Stallman and Sussman [1976], and London [1977]. Fikes' data base system records sets of support for deductions, and uses these in automatically erasing data derived from other erased data. His system does not use multiple justifications for single beliefs, and does not use the dependencies to control search. McDermott's earlier system also uses sets of supports as a data erasing mechanism, and allow multiple justifications for beliefs as well. These must lead to problems in his system, as no truth maintenance mechanism is used. McDermott's later system is somewhat more developed, but incorporates a number of extraneous forms of information into the recorded data dependencies, thereby obscuring the problems of truth maintenance and control. Stallman and Sussman's ARS electronics analysis system employs multiple sets of support and a monotonic truth maintenance system. ARS employs a mechanism for computing the sets of support of conditional proofs, but does not make these explicit as justifications. Instead of recording conditional proof justifications, the set of independent support is computed on the spot, and

never recomputed. ARS is a single-level system, and does not address the problem of separating levels of detail. London employs an extended Fikes-like dependency network in updating a simulation model.

Another class of systems each use mechanisms suggestive of dependency mechanisms. These systems include Hayes' [1975] planning system, Cox and Pietrzykowski's [1976] theorem prover, and Srinivasan's [1976] MDS. Hayes' system is organized so that each decision made is associated with the set of other decisions which influenced the making of the decision. When surprises during plan execution invalidate a choice, these records are used to erase all decisions influenced by the invalid decision. Procedural methods seem to be used to connect decisions with the underlying domain knowledge. Cox and Pietrzykowski's graphical deduction system builds a proof in graph structure. When backtracking leads to the removal of a unification from the graph, the unification histories are employed to erase only that part of the graph that depended on the removed unification. Srinivasan's system associates with each assertion a set of other assertions which were accessed in checking the consistency of the assertion with the existing data base.

Several systems employ forms of careful backtracking mechanisms. Stallman and Sussman's [1976] ARS system introduced the mechanism of dependency-directed backtracking. Katz and Manna [1976] use recorded dependencies between program invariants when an attempted proof of correctness fails. The dependencies are used first to search for a program statement to modify, and then to direct the updating of the other

invariants. Cox and Pietrzykowski's [1976] graphical deduction system analyzes the unification histories recorded in building a proof graph. If progress is halted, the analysis performed by the backtracking algorithm indicates a unification which, if discarded, will allow further progress in the search. Nevins [1974] presents a theorem prover which examines the proof graph when a splitting attempt fails. Berliner's [1974] chess program uses a perturbation technique (called the Causality Facility) to analyze the dependence features of the board situation on past moves. He also proposed the use of "lemmas" to record the reasons for ruling out possible moves. These lemmas are similar in nature to domain-specific NOGOODs, as they specify the conditions under which a move is bound to be bad. Latombe [1976] has indicated that his TROPIC system also performs clever backtracking.

One related mechanism is the context mechanism employed by CONNIVER [McDermott and Sussman 1974] and QA4 [Rulifson, Derksen and Waldinger 1973]. The basic objective of both context and truth maintenance systems is the ability to reason without confusion when using several mutually contradictory sets of beliefs. Such conflicting sets of beliefs arise in reasoning about sequences of actions and in reasoning about hypothetical assumptions. Context mechanisms are fortuitously useful for reasoning about sequences of actions. This is because action sequences generate trees of incrementally different situations, and context systems are structured into trees of incrementally different contexts. In addition to being similar in structure to the trees of situations arising from action sequences, contexts allow simultaneous access to distinct situations. This means that

the properties of objects in one situation can be compared to the properties of the objects in another. Because truth maintenance systems cannot easily inspect different situations, reasoning about actions in a truth maintenance system is awkward.

On the other hand, contexts are inappropriate for reasoning about hypothetical assumptions. Many hypotheticals are naturally independent. This allows changes of beliefs derived from one assumption to be unaffected by changes in beliefs in independent assumptions. Truth maintenance systems handle this easily. In context systems, hypothetical extensions of a context must be made in some particular order. This means that anomalous dependencies are unavoidable. Because of these anomalous dependencies, discarding one assumption by popping the corresponding context layer can result in discarding information derived from independent assumptions. This leads to the loss of useful information and wasted search efforts. Contexts can be viewed as approximating the logical dependencies between data. The problems introduced by the use of contexts are therefore avoided by using the logical dependencies themselves to compute beliefs, rather than the approximate relationships of context membership.

C. Future Work

You know my methods. Apply them.

Sir Arthur Conan Doyle, *The Sign of Four*

Many of the topics discussed in this report offer opportunities for future exploration and elaboration. Some of these concern the use of a truth maintenance system in explanation and hypothetical reasoning. Other problems are in improving the technical details of implementing truth maintenance systems. This section discusses these areas for future research.

A major application of the dependency relationships determined by a truth maintenance system is the explanation of computed entities in terms of the knowledge by which they were computed. The recorded justifications for beliefs only form the raw material from which explanations are to be constructed. The explanations produced by simple examinations of the justifications or foundations of beliefs are often cluttered with annoying details. Some of this unnecessary information can be removed by using the conditional proofs to restructure and summarize arguments. This report has indicated techniques for using this mechanism in structuring the knowledge of a program into hierarchical levels of detail, but much interesting work seems possible in developing techniques by which query routines can perform dynamic restructuring of arguments. These restructurings might take into account the knowledge available to the listener and the

purpose of the explanation. (Cf. [Carr and Goldstein 1977])

Several topics of interest concern domain-independent methods in hypothetical reasoning. As indicated in the discussion of backtracking, there are several possible criteria for analyzing the structure of assumptions involved in inconsistencies. Further exploration of backtracking schemes employing these criteria might provide added efficiencies in backtracking. Related topics include the use of the dependency relationships alone as measures of the strength or stability of arguments, and in estimating the effects of changes in beliefs.

A theoretical problem arising in the system is the development of a formal semantics for non-monotonic inference. Drew McDermott [personal communication] is developing one such semantics.

The power of dependency-directed backtracking calls for integration of this method with knowledge-based methods of hypothetical reasoning. Programs with knowledge of the semantics of nodes can greatly increase the efficacy of the knowledge-free automatic methods by supplying measures for the soundness of premises and arguments in backtracking and differential diagnosis. McDermott's [1974] TOPLE program, for instance, constructs what might be called abductive data dependencies to explain away inconsistencies. It would be interesting to investigate such methods in a system which also provided the deductive justifications presented here. Similarly, knowledge of the meanings

of the premises involved might be used in methods for generalizing the information derived from inconsistencies into more widely applicable rules.

On a technical level, there are several algorithmic improvements to be made in the basic truth maintenance process itself. Speedups in truth maintenance processing might be gained if a better understanding is developed of the best order for examination and the analysis of circularities is improved. Efficient (or even correct) methods for evaluating conditional proof justifications when their hypotheses are not valid and for switching between sets of hypotheses need to be developed.[Too Hard] The use of multiple supporting-justifications for beliefs is a topic for study, especially with respect to the impact of such a mechanism' on the other truth maintenance mechanisms of backtracking and conditional proofs. Truth maintenance is an incremental process in the sense that only those beliefs affected by changes are updated, but there is another sense of incrementality in which the effects of changes in assumptions are calculated only if actually necessary. Unfortunately, the only methods I know for implementing such a call-by-need truth maintenance system require examining all beliefs, not just those affected by changes. David McAllester [1977] has developed a truth maintenance system based on a representation using propositional clauses in conjunctive normal form. This representation allows several algorithms (such as truth maintenance and dependency-directed backtracking) to be unified into one algorithm.

One important problem is the detailing of methods integrating the use of dependencies with the sharing of data. Fahlman's [1977] NETL system avoids explosions of

computation space by using "virtual copies" of pieces of information. Straightforward schemes for using recorded justifications require explicit copies of the information, and cannot deal with shared structure. It would be very valuable to develop methods for this integration, perhaps of the nature of shared or virtually-copied dependency structures.

Finally, there are many interesting applications of dependencies to problems of control in problem solving systems. Explicit justifications allow the separation of the control and the knowledge embodied by the problem solver, since the problem solver can make derived knowledge depend only on non-control information. At the same time, dependencies permit explicit linking of control decisions to the information and decisions they are based on, opening the possibility for careful failure analysis by the problem solving interpreter. (Cf. [de Kleer, Doyle, Steele and Sussman 1977]) There are many problems for investigation in explicit reasoning about the justifications themselves (for instance, in reasoning about the existence of multiple proofs for some result).

Notes

AMORD

AMORD [de Kleer, Doyle, Rich, Steele and Sussman 1977] is a simple problem solving system devloped to illustrate the technique of using explicit control statements and dependencies in the control of reasoning. [de Kleer, Doyle, Steele and Sussman 1977]

NOGOOD

Records indicating sets of inconsistent assumptions were called NOGOODs in Stallman and Sussman's [1976] ARS.

Logics of Belief

Hintikka [1962] and other philosophers have made extensive studies of the logics of knowledge and belief. A logic of beliefs seemingly related to that used in a truth maintenance system is discussed by Belnap [1976].

THNOT

The non-monotonic assumption justification are the dependency system analogue of Micro-PLANNER's THNOT primitive. [Sussman, Winograd and Charniak 1971] Other related concepts are McCarthy and Hayes' [1969] CONSISTENT predicate, Sandewall's [1972] UNLESS predicate, and McDermott's [1977] PRESUMABLY. The non-monotonic assumption justifications have an advantage over these systems in that the nature of the assumption is made explicit and accessible in future deductions. This allow data derived from an assumption to be automatically discarded when new information overrides the assumption.

NEEDCHOICE

ARS [Stallman and Sussman 1976] represents the reasons for making assumptions as NEEDCHOICE assertions. Since ARS uses a monotonic dependency system, procedural mechanisms are necessary to connect these NEEDCHOICE assertions with the assumptions they control during backtracking and truth maintenance.

Propagation

Equivalence classes arise naturally when using methods which propagate information through a fixed knowledge structure. A good example is the EL electronic circuit analysis program. [Stallman and Sussman 1976] This program makes a fundamental use of coincidences and contradictions between voltages and currents in its method of the propagation of constraints.

Interfaces

The mechanism for separating levels of detail also applies to interfaces between independent systems. In these interfaces, each side of the boundary looks like a call on the

other side. Such interfaces can be described by using conditional proofs to transmit information in both directions. This requires that the names of each of the sides of the boundary to be attached to the boundary.

Slices

Slices [Sussman 1977] are a way of representing multiple overlapping views of objects. They provided a major source of the motivation for developing our methods of separating levels of detail in explanations.

Complexity

Part of the abstract problem of truth maintenance is to find an assignment of states to nodes consistent with a set of logical constraints relating these states. This is reminiscent of the problem of finding a satisfying assignment of values to variables in a propositional formula. It is easy to construct a correspondence between truth maintenance and the satisfaction of propositional formulas. I attempted to use such a correspondence to show that the problem faced by truth maintenance was NP-complete. [Cook 1971, Karp 1972] I failed because the set of constraints relating beliefs is guaranteed to be satisfiable (since unsatisfiable constraints are ruled out as program bugs). This foiled my attempts to invoke the NP-completeness of problems like CNF-SAT and DNF-UNSAT in my proof.

Efficiency

There are several variations on the algorithm for truth maintenance presented here. The major dimension for variation concerns the order in which the graph of nodes and justifications is searched. Our algorithm uses a depth-first search. Other orders are possible. The most desirable would be one which always found support for the node being searched, and determined the set of nodes involved in circularities with as little trouble as possible.

Analysis

The efficiency of the relaxation process might be improved by using some form of analysis of circularities. For instance, the set of nodes left unsupported by the first step of truth maintenance can be analyzed as a graph to find the strongly connected components. These components are related by a partial order. The most efficient procedure would to determine support for nodes in the components which are minimal in this partial order, since these might provide support for node in non-minimal components.

FINDINDEP

This procedure was given the name FINDINDEP in Stallman and Sussman's [1976] ARS system.

PMMCS

Nicholas Rescher [1964] presents a solution to the problem of counterfactual conditionals based on the concept of Preferred Maximal Mutually Compatible Subsets of

beliefs. His method partitions the set of all beliefs into compatible subsets of beliefs. Some of these sets may be preferable to others. For instance, if an inconsistency involves an experimental hypothesis and a law of nature, it is preferable to surrender the experimental hypothesis (as in "Back to the drawing board."). If such preferences were available to the dependency-directed backtracking system, it would be possible to retract premises as well as assumptions. On the other hand, one can argue that the partial order discrimination used in selecting assumptions for retraction is in fact a dynamically computed preference measure on the different *out* nodes involved in the inconsistency.

Combinatorics

Dependency-directed backtracking offers two main sources of efficiency over chronological backtracking methods, such as that used in Micro-PLANNER. [Sussman, Winograd and Charniak 1971] One advantage is that irrelevant choices made chronologically later than a faulty choice are ignored. This saves the system from considering many useless combinations of choices. In addition, it allows deductions made on the basis of faulty choices to be distinguished from those stemming from irrelevant choices. This means that the truth maintenance system need only discard those deductions based on retracted assumptions. Another advantage of dependency-directed backtracking is in the summarization of the inconsistency of certain sets of assumptions. Because these summarizations are valid beyond the existence of the inconsistency, the system automatically prevents any future set of assumptions from including the inconsistent set as a subset. This leads to the avoidance of more pointless combinations of assumptions. The power of dependency-direction is illustrated by Stallman and Sussman [1976], who present an example in which traditional methods of backtracking would consider a number of states equal to the the product of the sizes of the independent choice-sets. The use of dependency-direction reduces the number of states considered to the sum of the sizes of the independent choice-sets. G. J. Sussman has informed me of a specific example of interest. In a particular six transistor circuit, dependency-directed backtracking reduces the number of contradictions to only 2. This is using a standard heuristic order when choosing transistor states. Traditional backtracking methods might consider 3**6 = 729 states in this example. The number of contradictions increases to only 13 when the heuristically worst choice is made in each decision.

Too Hard

The problem of evaluating the validity of conditional proofs may be too hard in general. If the hypotheses of the conditional proof are not valid, that is, if the *in*hypotheses are not *in* and the *out*hypotheses are not *out*, the evaluation requires switching beliefs so that the hypotheses are valid. This may require recursive truth maintenance, which in turn may call for further evaluation of conditional proof justifications. I have no algorithms for this process. It may be that the difficulty of this problem requires that such conditional proofs be left to the problem solver, reserving only the simple cases for the truth maintenance system.

References

[Belnap 1976]
N. D. Belnap, "How a Computer Should Think," in *Contemporary Aspects of Philosophy*, Gilbert Ryle, ed., Stocksfield: Oriel Press, 1976.

[Berliner 1974]
Hans J. Berliner, "Chess as Problem Solving: The Development of a Tactics Analyzer," CMU Computer Science Department, March 1974.

[Carr and Goldstein 1977]
Brian Carr and Ira Goldstein, "Overlays: A Theory of Modelling for Computer-Aided Instruction," MIT AI Lab, Memo 406, February 1977.

[Cook 1971]
S. A. Cook, "The Complexity of Theorem-Proving Procedures," Proc. 3rd Annual ACM Symp. on the Theory of Computing, 1971, pp. 151-158.

[Cox and Pietrzykowski 1976]
Philip T. Cox and T. Pietrzykowski, "A Graphical Deduction System," Department of Computer Science Research Report CS-75-35, University of Waterloo, July 1976.

[de Kleer, Doyle, Rich, Steele and Sussman 1977]
Johan de Kleer, Jon Doyle, Charles Rich, Guy L. Steele Jr., and Gerald Jay Sussman, "AMORD: A Deductive Procedure System," MIT AI Lab, Memo 435, September 1977.

[de Kleer, Doyle, Steele and Sussman 1977]
Johan de Kleer, Jon Doyle, Guy L. Steele Jr., and Gerald Jay Sussman, "Explicit Control of Reasoning," MIT AI Lab, Memo 427, June 1977.

[Fahlman 1977]
Scott E. Fahlman, "A System for Representing and Using Real World Knowledge," forthcoming MIT PHD thesis.

[Fikes 1975]
Richard E. Fikes, "Deductive Retrieval Mechanisms for State Description Models," *IJCAI4*, September 1975, pp. 99-106.

[Hayes 1975]
Philip J. Hayes, "A Representation for Robot Plans," *IJCAI4*, September 1975, pp. 181-188.

[Hintikka 1962]
Jaakko Hintikka, *Knowledge and Belief*, Cornell University Press, Ithica, New York, 1962.

[Karp 1972]
R. M. Karp, "Reducibility Among Combinatorial Problems," in Miller and Thatcher, editors, *Complexity of Computer Computations*, Plenum Press, New York, 1972, pp. 85-104.

[Katz and Manna 1976]
Shmuel Katz and Zohar Manna, "Logical Analysis of Programs," *Comm. ACM*, Vol. 19, No. 4, pp. 188-206.

[Latombe 1976]
Jean-Claude Latombe, "Artificial Intelligence in Computer-Aided Design: The TROPIC System," SRI AI Center, TN-125, February 1976.

[London 1977]
Phil London, "A Dependency-Based Modelling Mechanism for Problem Solving," Computer Science Department TR-589, University of Maryland, November 1977.

[McAllester 1977]
David A. McAllester, "Implementing Truth Maintenance Systems with Bidirectional Disjunctive Clauses," MIT EE&CS Bachelor's Thesis proposal, October 1977.

[McCarthy and Hayes 1969]
J. McCarthy and P. J. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence," in Meltzer and Michie, *Machine Intelligence 4*, pp. 463-502.

[McDermott 1974]
Drew Vincent McDermott, "Assimilation of New Information by a Natural Language-Understanding System," MIT AI Lab, AI-TR-291, February 1974.

[McDermott 1975]
Drew V. McDermott, "Very Large PLANNER-type Data Bases," MIT AI Lab, AI Memo 339, September 1975.

[McDermott 1977]
Drew V. McDermott, "Flexibility and Efficiency in a Computer Program for Designing Circuits," MIT AI Lab, TR-402, June 1977.

[McDermott and Sussman 1974]
Drew V. McDermott and Gerald Jay Sussman, "The CONNIVER Reference Manual," MIT AI Lab, AI Memo 259a, January 1974.

[Nevins 1974]
Arthur J. Nevins, "A Human-Oriented Logic for Automatic Theorem Proving," *JACM 21*, #4 (October 1974), pp. 606-621.

[Rescher 1964]
N. Rescher, *Hypothetical Reasoning*, Amsterdam: North Holland 1964.

[Rulifson, Derksen and Waldinger 1973]
Johns F. Rulifson, Jan A. Derksen, and Richard J. Waldinger, "QA4: A Procedural Calculus for Intutitive Reasoning," SRI AI Center, Technical Note 73, November 1973.

[Sandewall 1972]
E. Sandewall, "An Approach to the Frame Problem, and its Implementation," *Machine Intelligence 7*, pp. 195-204, 1972.

[Srinivasan 1976]
Chitoor V. Srinivasan, "The Architecture of Coherent Information System: A General Problem Solving System," *IEEE Transactions on Computers*, Vol. C-25, No. 4, April 1976, pp. 390-402.

[Stallman and Sussman 1976]
Richard M. Stallman and Gerald Jay Sussman, "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," MIT AI Memo 380, September 1976. Revised version published in *Artificial Intelligence*, Vol. 9, No. 2, (October 1977), pp. 135-196.

[Sussman 1977]
Gerald Jay Sussman, "Slices: At the Boundary Between Analysis and Synthesis," MIT AI Lab, Memo 433, July 1977.

[Sussman, Winograd and Charniak 1971]
Gerald Jay Sussman, Terry Winograd and Eugene Charniak, "MICRO-PLANNER Reference Manual," MIT AI Lab, AI Memo 203a, December 1971.

Appendix 1. A TMS Glossary

Aldiborontiphoscophornio!
Where left you Chrononhotonthologos?

Henry Carey, *Chrononhotonthologos*

The AFFECTED-CONSEQUENCES of a node are those nodes whose current support-status rests on the node; precisely, those consequences of the node such that the node is a supporting-node of each of these consequences.

The ANCESTORS of a node are those nodes involved at some level in determining the current support-status of the node; precisely, the transitive closure of the set of supporting-nodes of the node under the operation of taking supporting-nodes.

ANOMALOUS CHRONOLOGICAL DEPENDENCIES are present when beliefs depend unnecessarily on other, logically independent beliefs. A good example of such anomalous dependencies are those seen in Micro-PLANNER, where each assertion depends on all chronologically previous choices, regardless of considerations of the logical relations between the assertion and the choices.

The ANTECEDENTS of a node are those nodes which currently support belief in the node. The antecedents of a node are the same as the SUPPORTING-NODES if the node is *in*. Nodes which are *out* have no antecedents.

ASSUMPTIONS are nodes justified by non-monotonic justifications. The addition of new knowledge can cause the justifications for belief in assumptions to become invalid. Specifically, an assumption is a node believed on the basis of a lack of knowledge about some other belief. A typical form of an assumption is a node F whose justification is the predicate (OUT $\sim F$), where $\sim F$ denotes the negation of F. In this case, belief in F will be justified as long as contradictory knowledge is not present.

BACKTRACKING is the process of undoing a failure or inconsistency by retracting some premise or assumption underlying the failure or inconsistency.

BELIEF in a node results from knowledge of a valid justification.

CHRONOLOGY describes the dependence of actions on the temporal ordering of their execution.

CIRCULARITIES can occur in dependency relationships. These circularities require special processing during truth maintenance.

CONDITIONAL PROOFS are justifications which support a belief if a specified belief (the consequent of the conditional proof) is believed when given specified hypotheses (the hypotheses of the conditional proof).

The CONSEQUENCES of a node are those nodes such that the node occurs in an justification of each consequence.

A CONTRADICTION is a node which has been designated as an inconsistency to the truth maintenance system. The backtracking mechanisms attempts to ensure that no contradiction is believed.

The CP-CONSEQUENT-LIST of a node contains those nodes possessing a CP-JUSTIFICATION with the node as the consequent of the conditional proof.

A DEDUCTION is a type of justification for belief in a node in which belief in the node is justified if each node in a designated set of nodes is believed.

DEPENDENCIES are relationships between beliefs. The most generally useful dependencies in a truth maintenance system are the relations of ANTECEDENTS, CONSEQUENCES, SUPPORTING-NODES and AFFECTED-CONSEQUENCES.

DEPENDENCY-DIRECTED describes processes, such as DEPENDENCY-DIRECTED BACKTRACKING, which operate on beliefs by searching through the nodes related by dependencies to the beliefs.

FINDINDEP is a procedure for determining the set of nodes which are the independent support of a belief relative to a specified set of beliefs. This procedure is the basis for mechanisms dealing with summarization by conditional proofs.

The FOUNDATIONS of a node are those nodes involved in the well-founded support for belief in the node; precisely, the transitive closure of the antecedents of the node under the operation of taking antecedents.

A HYPOTHESIS is an underived belief, that is, either a PREMISE or an ASSUMPTION. Hypotheses of conditional proof justifications need not be premises or assumptions -- they are just treated as such for the purposes of the conditional proof.

IN describes the condition of a node which is believed to be *true* due to knowledge of a valid justification supporting this belief.

An JUSTIFICATION of a node is a representation of a reason for belief in the node. There are two basic representations used in the truth maintenance system presented here, called support-list justifications (SL-JUSTIFICATIONS) and conditional-proof justifications (CP-JUSTIFICATIONS).

The JUSTIFICATION-SET of a node is the set of justifications for belief in the node.

MONOTONIC TRUTH MAINTENANCE SYSTEMS are systems in which belief in a node cannot be predicated upon a lack of belief in some other node.

A NODE is the fundamental entity to which justifications for belief can be attached.

A NOGOOD is a node summarizing the independent support of a contradiction relative to the set of assumptions underlying the contradiction.

A NOGOOD-SET is a set of assumptions designated to be inconsistent by a NOGOOD.

NON-CHRONOLOGICAL describes processes in which the order of actions does not affect the results, so that the relationships between the data produced can be summarized in time-independent, logical terms.

OUT describes the condition of a node for which no valid justifications are known.

A PREMISE is a belief which depends on no other beliefs.

The REPERCUSSIONS of a node are those other nodes whose support-statuses are affected at some level by the support-status of the node; precisely, the transitive closure of the affected-consequences of the node under the operation of taking affected-consequences.

A SL-JUSTIFICATION consists of a two lists of nodes, the *in*list and the *out*list. It is valid if each node in the *in*list is *in*, and each node in the *out*list is *out*.

The STATUS of a node normally refers to the SUPPORT-STATUS of the node.

The SUPPORT for a belief is a generic term and may refer to the SUPPORTING-JUSTIFICATION, ANTECEDENTS, SUPPORTING-NODES, FOUNDATIONS or ANCESTORS of the node, depending on the context of usage.

The SUPPORT-STATUS of a node is another name for the *in*ness or *out*ness of the node.

The SUPPORTING-JUSTIFICATION determines a proof of belief in the node in terms of other nodes with well-founded (non-circular) support.

The SUPPORTING-NODES of a node is the set of nodes affecting the current support-status of the node.

TRUTH MAINTENANCE refers to the process by which beliefs are redetermined when other beliefs change due to the addition of new information or the retraction of premises.

TRUTH MAINTENANCE SYSTEMS are systems in which beliefs are determined from recorded logical justifications.

UNSATISFIABLE DEPENDENCY RELATIONSHIPS are cycles of justifications such that no assignment of *in* or *out* to the nodes involved is consistent with the justifications recorded for belief in the nodes.

VALID JUSTIFICATIONS are justifications which as predicates evaluate to *true*.

WELL-FOUNDED SUPPORT for a belief is an argument in terms of the recorded justifications from the basic premises and assumptions of the system upwards with no cycles in the arguments.

Appendix 2. Monotonic Truth Maintenance Systems

In monotonic truth maintenance systems, no belief can depend upon other nodes being *out*. With this restriction, the *in*ning of a node cannot cause the *out*ing of another. Truth maintenance processing is simpler to implement in a monotonic system than in the non-monotonic system described in the text. The monotonic system also has many of the same uses and general properties. The basic limitation of the monotonic dependency system is its inability to model assumptions. The lack of non-monotonic justifications means that procedural mechanisms for making assumptions must be used. These procedural devices must be able to handle all the problems involving circularities that are exhibited in the non-monotonic system. The procedural mechanisms must be able to effect non-monotonic inferences, and connect assumptions with the reasons for being made.

The representation of justifications can be simplified somewhat in a monotonic truth maintenances system. A SL-justification becomes a single list of nodes. CP-justifications have only one list of hypotheses. The concepts of consequences, supporting-justifications, supporting-nodes, ancestors and repercussions are as in the non-monotonic case. In a monotonic system, the list of supporting-nodes of a node is identical to the list of nodes in the supporting-justification.

The process of truth maintenance is simplified somewhat in a monotonic system. It is conveniently divided into two processes; *out*ing and un*out*ing. The process of *out*ing occurs whenever a node is changed from *in* to *out*. It proceedes by {1} making a list of the invoking node and its repercussions, {2} setting the support-status of each of these nodes to *out*, {3} removing the supporting-justifications from each of the nodes. After these steps have been performed for each of the nodes in the list, each of the nodes is un*out*ed.

The un*out*ing of a node proceeds by examining the justification-set of the node for an SL-justification containing only *in* nodes. If such an justification is found, it is made the supporting-justification of the node, the node's support-status becomes *in*, and all *out* consequences of the node are un*out*ed recursively. The handling of CP-justifications is essentially unchanged from that described previously for non-monotonic systems.

The major effect of monotonicity is that the mechanisms for making and maintaining assumptions must be done by means external to the truth maintenance system. One method for effecting non-monotonic relationships is that of using forget and recall functions, which are functions attached to the node. A forget function is run whenever the attached node is *out*ed, and a recall function is run whenever the attached node is *in*ned. To implement the assumption of a node F using such functions, a forget function can be attached to $\sim F$ such that if $\sim F$ is *out*ed, F will be made *in* (as a premise), along with a recall function on $\sim F$ which will *out* F (by retracting it) if $\sim F$ is *in*ned. In addition, these functions should also check to make sure that the node representing the reason for making

the assumption is *in*. The use of a mechanism like this dangerous. Because functions are being used instead of explicit justifications, the recognition and handling of circularities and inconsistencies are obscured. Because of the chronological nature of these functions, no guarantee can be made that all feasible selections of statuses can be made in the event of circularities. Also, unsatisfiable circularities in the dependency structures may be undetectable, ensuring an infinite loop of assuming, un*out*ing, *out*ing, and reassuming.

Many of the applications of the general truth maintenance system are also possible using a monotonic system. The uses of dependencies in explanation, generalization and separation of levels of detail, and in dependency-directed backtracking are similar to those in the non-monotonic system. Backtracking is affected by the fact that assumptions can not be made through the use of the dependencies. Because of this, each node representing an assumption must be explicitly marked as such so that the backtracker can recognize it as an assumption. In addition, while nogoods are representable as before, the ruling out of inconsistent sets of assumptions cannot be done by means of new justifications, but must operate by external mechanisms.

Appendix 3. An Implementation of a TMS

This appendix presents the February, 1978 version of the TMS. A set of descriptions of the functions are provided, followed by the MacLISP programs.

TMS-MAKE-NODE -- (TMS-MAKE-NODE <external-name>)

This function creates a new TMS-node with a given name.

TMS-SL-JUSTIFY -- (TMS-SL-JUSTIFY <node> <insupporters> <outsupporters> <argument>)

This function gives a TMS node a new support-list type justification, which is valid if each of the nodes of the *in*supporters list is *in*, and each of the nodes of the *out*supporters list is *out*. The argument is an uninterpreted slot used to record the external form of the justification, and is retrievable via the TMS-JUSTIFICATION-ARGUMENT function described below.

TMS-CP-JUSTIFY

-- (TMS-CP-JUSTIFY <node> <consequent> <inhypotheses> <outhypotheses> <argument>)

This gives a TMS node a new justification which is valid if, when the *in*hypotheses are *in* and the *out* hypotheses are *out*, the consequent node is believed. As in TMS-SL-JUSTIFY, the argument is an uninterpreted record of the external form of the justification.

TMS-PROCESS-CONTRADICTION

-- (TMS-PROCESS-CONTRADICTION <name> <node> <type> <contradiction-function>)

This declares a TMS node to represent a contradiction. The name and type are uninterpreted mnemonics provided by the external system to be printed out during backtracking. The contradiction-function, if supplied, should be a LISP function to be called with the contradiction node as its argument when the backtracker can find no backtrackable choicepoints.

TMS-SUPPORT-STATUS -- (TMS-SUPPORT-STATUS <node>)

This function returns the support-status, either 'IN or 'OUT, of a node.

TMS-JUSTIFICATIONS -- (TMS-JUSTIFICATIONS <node>)

This function returns the list of justifications of the node. This list contains both the support-list and conditional-proof justifications attached to the node.

TMS-SUPPORTING-JUSTIFICATION -- (TMS-SUPPORTING-JUSTIFICATION <node>)

This function returns the current justification of the node.

TMS-JUSTIFICATION-ARGUMENT -- (TMS-JUSTIFICATION-ARGUMENT <justification>)

This function returns the external argument associated with the given justification.

TMS-ANTECEDENTS -- (TMS-ANTECEDENTS <node>)

This function returns the list of nodes determining well-founded support for the given node. This list is extracted from the supporting-justification if the node is *in*, and is empty if the node is *out*.

TMS-CONSEQUENCES -- (TMS-CONSEQUENCES <node>)

This function returns the list of nodes whose list of antecedents mentions the given node.

TMS-EXTERNAL-NAME -- (TMS-EXTERNAL-NAME <node>)

This function returns the user-supplied name of a node.

TMS-IS-IN -- (TMS-IS-IN <node>)

This predicate is true iff the node is *in*.

TMS-IS-OUT -- (TMS-IS-OUT <node>)

This predicate is true iff the node is *out*.

TMS-RETRACT -- (TMS-RETRACT <node>)

This function will remove all premise-type justifications from the set of justifications of the node.

TMS-PREMISES -- (TMS-PREMISES <node>)

This function returns a list of the premises among the foundations of the node.

TMS-ASSUMPTIONS -- (TMS-ASSUMPTIONS <node>)

This function returns a list of the assumptions among the foundations of the node.

TMS-INSTALL-SIGNAL-FORGETTING-FUNCTION

-- (TMS-INSTALL-SIGNAL-FORGETTING-FUNCTION <node> <fun>)

This function sets the LISP function that the TMS will use to signal the changing of the support-status of the node from *in* to *out*. When such a change occurs, the supplied function will be called with the external name of the node as its argument.

TMS-INSTALL-SIGNAL-RECALLING-FUNCTION

-- (TMS-INSTALL-SIGNAL-RECALLING-FUNCTION <node> <fun>)

This function sets the LISP function that the TMS will call with the node's external name as its argument when changing the support-status of the node from *out* to *in*.

The TMS also generates new "facts" internally during backtracking. These will therefore occur in explanations and antecedents of the nodes requested and justified by the external systems. The internal facts generated by the TMS are atomic symbols with certain properties. The following functions are provided to manipulate these internal facts.

TMS-FACTP -- (TMS-FACTP <thing>)
This predicate is true iff the thing is an internal TMS fact.

TMS-FACT-NODE -- (TMS-FACT-NODE <fact>)
This function returns the TMS node associated with an internal fact.

TMS-FACT-STATEMENT -- (TMS-FACT-STATEMENT <fact>)
This function returns the symbolic statement of the meaning of an internal fact. This statement refers to the external names of the other facts, such as contradictions and assumptions, which were involved in the making of the fact.

The following two functions are supplied for debugging purposes.

TMS-INIT -- (TMS-INIT)
This function clears the state of the TMS by resetting all internal variables and clearing all properties and internings of TMS nodes.

The TMS has the following switches which may be set for wallpaper purposes.

Variable (Default value)

TMS-SEE-TMP-SW (NIL)
Announces truth maintenance processing.

TMS-SEE-TMP-INVOKER-SW (T)
Controls printing of nodes invoking truth maintenance processing if *TMS-SEE-TMP-SW* is set.

TMS-SEE-JUSTIFY-SW (NIL)
Announces the addition of a new justification for a node.

TMS-SEE-CONTRADICTIONS-SW (T)
Announces the processing of a contradiction.

The program as follows uses several macros.

```
(LET ((var1 init1) --- (varn initn)) body)
```

is equivalent to

```
((LAMBDA (var1 --- varn) body) init1 --- initn).
```

```
(EQCASE exp (val1 body1) --- )
```

is effectively equivalent to

```
(COND ((eq exp 'val1) body1) ---).
```

The last clause in an EQCASE may begin with the value ELSE, which forms a catch-all clause at the end of the COND. The macro-character " quotes the following form, substituting in the values of any forms preceded by the macro-character , and inserting as a list segment the value of any form preceded by the macro character @.

```
001
002   (COMMENT TRUTH MAINTENANCE SYSTEM GLOBAL VARIABLES)
003
004
005   ;;; THIS DECLARATION SPECIFIES ALL OF THE SWITCHES NECESSARY TO COMPILE THE TMS.
006   ;;; THESE INCLUDE THE LOADING OF A FILE OF MACROS USED IN THE TEXT, AND DECLARATIONS
007   ;;; OF SOME LISP FUNCTIONS USED BY THE PROGRAM.
008
009   (DECLARE (MAPEX T)                                    ;OPEN CODE LOOPS
010            (EXPR-HASH T)                                ;ENABLE RECOMPILATION HACKS
011            (MACROS NIL)                                 ;RETAIN NO MACRO DEFINITIONS
012            (FASLOAD LINIT FASL DSK AMORD)               ;LOAD IN MACROS
013            (SPECIAL BASE *NOPOINT)
014            (*FEXPR GCTWA))
015
016   ;;; THIS DECLARATION SPECIFIES ALL OF THE GLOBAL VARIABLES USED IN THE TMS.
017   ;;; *TMS-NOTED-IN-NODES* IS USED DURING TRUTH MAINTENANCE TO ACCUMULATE A LIST
018   ;;; OF ALL NODES EXAMINED BY TRUTH MAINTENANCE WHICH WERE IN UPON ENTRY TO
019   ;;; THE EXAMINATION PROCESS.
020   ;;; *TMS-NOTED-OUT-NODES* ACCUMULATES ENTERING OUT NODES DURING TRUTH MAINTENANCE.
021   ;;; *TMS-PROCESS-QUEUE* IS USED TO QUEUE UP NODES FOR EXAMIMINATION FOR WELL-FOUNDED
022   ;;; SUPPORT DURING TRUTH MAINTENANCE.
023   ;;; *TMS-CONTRADICTION-ASSUMPTIONS* IS USED DURING BACKTRACKING TO ACCUMULATE AN ALIST
024   ;;; OF ASSUMPTION NODES AND THE LISTS OF OUT NODES THEY DEPEND UPON.
025   ;;; *TMS-FINDINDEP-IN-LIST* ACCUMULATES IN NODES DURING TMS-FINDINDEP PROCESSING.
026   ;;; *TMS-FINDINDEP-OUT-LIST* ACCUMULATES OUT NODES DURING TMS-FINDINDEP PROCESSING.
027   ;;; *TMS-SEE-JUSTIFICATIONS-SW* CONTROLS WALLPAPER PRINTING DURING THE TMS-XX-JUSTIFY
028   ;;; FUNCTIONS.
029   ;;; *TMS-SEE-TMP-SW* CONTROLS WALLPAPER PRINTING DURING THE MAIN TRUTH MAINTENANCE FUNCTION.
030   ;;; *TMS-SEE-TMP-INVOKER-SW* CONTROLS WALLPAPER PRINTING DURING THE MAIN TRUTH MAINTENANCE
031   ;;; FUNCTION OF WHICH NODE CAUSED TRUTH MAINTENANCE.
032   ;;; *TMS-SEE-CONTRADICTIONS-SW* CONTROLS WALLPAPER PRINTING OF BACKTRACKING DUE TO
033   ;;; CONTRADICTIONS.
034   ;;; *TMS-SEE-CULPRITS-SW* CONTROLS WALLPAPER PRINTING OF THE ASSUMPTIONS UNDERLYING
035   ;;; CONTRADICTIONS DURING BACKTRACKING.
036   ;;; *TMS-GENS* STORES THE CURRENT GENSYM NUMBER.
037   ;;; *TMS-GENS-LIST* ACCUMULATES A LIST OF ALL SYMBOLS GENSYMED BY TMS-GENS.
038   ;;; *TMS-INTERN-SW* CONTROLS THE INTERNING OF GENSYMED SYMBOLS MADE BY TMS-GENS.
039
040   ;;; STIMULATE IS A FUNCTION CALLED WHEN A NODE GOES FROM IN TO OUT DURING TRUTH MAINTENANCE.
041   ;;; IT IS GIVEN THE EXTERNAL NAME OF THE NODE AS ITS ARGUMENT.
042   ;;; DESTIMULATE IS A FUNCTION CALLED WHEN A NODE GOES FROM OUT TO IN DURING TRUTH MAINTENANCE.
043   ;;; IT IS GIVEN THE EXTERNAL NAME OF THE NODE AS ITS ARGUMENT.
044
045   (DECLARE (SPECIAL *TMS-NOTED-IN-NODES*
046                     *TMS-NOTED-OUT-NODES*
047                     *TMS-PROCESS-QUEUE*
048
049                     *TMS-CONTRADICTION-ASSUMPTIONS*
050
051                     *TMS-FINDINDEP-IN-LIST*
052                     *TMS-FINDINDEP-OUT-LIST*
053
054                     *TMS-SEE-TMP-SW*
055                     *TMS-SEE-TMP-INVOKER-SW*
056                     *TMS-SEE-JUSTIFICATIONS-SW*
057                     *TMS-SEE-CONTRADICTIONS-SW*
058                     *TMS-SEE-CULPRITS-SW*
059
060                     *TMS-GENS*
061                     *TMS-GENS-LIST*
062                     *TMS-INTERN-SW*)
063
064             (*EXPR STIMULATE
065                    DESTIMULATE))
```

```
001
002   ;;; THIS FUNCTION IS THE MEANS BY WHICH THE TMS GENERATES NEW SYMBOLS.
003   ;;; THE SYMBOLS GENERATED ARE RECORDED ON A LIST TO FACILITATE REINITIALIZATION.
004   ;;; NORMALLY, THE SYMBOLS ARE NOT INTERNED.
005   ;;; BASE AND *NOPOINT ARE SPECIAL LISP VARIABLES.
006   ;;; THEIR REBINDING AVOIDS PROBLEMS DUE TO USER SETTINGS OF THESE VARIABLES.
007
008   (DEFUN TMS-GENS (X)
009          (SETQ *TMS-GENS* (1+ *TMS-GENS*))
010          ((LAMBDA (NAME)
011                   (AND *TMS-INTERN-SW* (INTERN NAME))
012                   (SETQ *TMS-GENS-LIST* (CONS NAME *TMS-GENS-LIST*))
013                   NAME)
014           (MAKNAM (APPEND '(T M S -)
015                           (AND X (NCONC (EXPLODEC X) '(-)))
016                           ((LAMBDA (BASE *NOPOINT)
017                                    (EXPLODEC *TMS-GENS*))
018                            8. T)))))
019
020   (SETQ *TMS-GENS* 0)
021   (SETQ *TMS-GENS-LIST* NIL)
022   (SETQ *TMS-INTERN-SW* NIL)
023
024   ;;; THESE INITIALIZE THE SYSTEM SWITCHES CONTROLLING WALLPAPER PRINTING.
025
026   (SETQ *TMS-SEE-TMP-SW* NIL)
027   (SETQ *TMS-SEE-TMP-INVOKER* NIL)
028   (SETQ *TMS-SEE-JUSTIFICATIONS-SW* NIL)
029   (SETQ *TMS-SEE-CONTRADICTIONS-SW* T)
030   (SETQ *TMS-SEE-CULPRITS-SW* NIL)
031
032   ;;; THIS FUNCTION INITIALIZES THE INTERNAL STATE OF THE TMS.
033   ;;; IT CAN BE CALLED ANY NUMBER OF TIMES.
034   ;;; AFTER IT IS CALLED, NO TMS DATA STRUCTURES SHOULD REMAIN
035   ;;; EXCEPT THOSE POINTED TO BY USER VARIABLES.
036
037   (DEFUN TMS-INIT ()
038          (SETQ *TMS-NOTED-IN-NODES* NIL)
039          (SETQ *TMS-NOTED-OUT-NODES* NIL)
040          (MAPC '(LAMBDA (G) (MAKUNBOUND G) (SETPLIST G NIL) (REMOB G))
041                *TMS-GENS-LIST*)
042          (SETQ *TMS-GENS-LIST* NIL)
043          (SETQ *TMS-GENS* 0)
044          (GCTWA T)
045          'DONE)
046
047   (TMS-INIT)
```

```
001
002   (COMMENT TRUTH MAINTENANCE DATA STRUCTURES)
003
004   ;;; TMS NODES HAVE THE FOLLOWING ATTRIBUTES KEPT IN A SPECIAL DATA STRUCTURE:
005
006   ;;; TMS-FINDINDEP-MARK
007   ;;; THIS IS A BIT USED TO INDICATE WHETHER THE NODE IN QUESTION HAS BEEN
008   ;;; EXAMINED BY THE TMS-FINDINDEP SWEEP PHASE.
009
010   ;;; TMS-SUBORDINATES-MARK
011   ;;; THIS BIT IS USED IN THE UPWARD PHASE OF THE TMS-FINDINDEP PROCESS
012   ;;; TO MARK EACH NODE WHICH HAS ANY OF THE HYPOTHESES OF THE CONDITIONAL
013   ;;; PROOF OR THEIR CONSEQUENCES AMONG ITS SUPPORTERS.
014
015   ;;; TMS-SUPERIORS-MARK
016   ;;; THIS IS A THREE-WAY INDICATOR USED DURING BACKTRACKING TO SHOW WHETHER THE
017   ;;; NODE HAS ASSUMPTIONS AMONG ITS CONSEQUENCES, DOES NOT HAVE SUCH CONSEQUENT
018   ;;; ASSUMPTIONS, OR HAS NOT BEEN EXAMINED YET.
019
020   ;;; TMS-TMP-MARK
021   ;;; THIS BIT SHOWS WHETHER THE NODE IS QUEUED UP TO BE EXAMINED DURING TRUTH
022   ;;; MAINTENANCE PROCESSING.  IT SAVES DOING A MEMQ DOWN A FREQUENTLY LARGE LIST.
023
024   ;;; TMS-NOTED-MARK
025   ;;; THIS BIT SHOWS WHETHER THE NODE HAS BEEN REACHED BY THE CURRENT INVOCATION
026   ;;; OF TRUTH MAINTENANCE.  IT SAVES DOING A MEMQ DOWN ANOTHER FREQUENTLY LARGE LIST.
027
028   ;;; TMS-SUPPORT-STATUS
029   ;;; THIS THREE-WAY INDICATOR SHOWS WHETHER THE NODE IS IN, OUT, OR IS STILL BEING
030   ;;; EXAMINED (NIL).
031
032   ;;; TMS-SL-JUSTIFICATIONS
033   ;;; THIS IS A LIST OF ALL OF THE SUPPORT-LIST JUSTIFICATIONS POSSESSED BY THE NODE.
034
035   ;;; TMS-CP-JUSTIFICATIONS
036   ;;; THIS IS A LIST OF ALL OF THE CONDITIONAL-PROOF JUSTIFICATIONS POSSESSED BY THE NODE.
037
038   ;;; TMS-SUPPORTING-JUSTIFICATION
039   ;;; THIS IS A POINTER TO THE CURRENTLY SUPPORTING JUSTIFICATION IF THE NODE IS IN,
040   ;;; OR NIL IF THE NODE IS OUT.  THE POINTER IS ALWAYS TO A SUPPORT-LIST JUSTIFICATION,
041   ;;; NEVER TO A CONDITIONAL PROOF JUSTIFICATION.  CP-JUSTIFICATIONS ARE CONVERTED TO
042   ;;; SL-JUSTIFICATIONS USING TMS-FINDINDEP.
043
044   ;;; TMS-SUPPORTING-NODES
045   ;;; THIS IS A LIST OF ALL NODES USED IN DETERMINING THE STATUS (IN OR OUT) OF THE NODE.
046   ;;; IF THE NODE IS IN, IT IS JUST THE LIST OF ALL NODES MENTIONED IN THE SUPPORTING
047   ;;; JUSTIFICATION OF THE NODE.  IF THE NODE IS OUT, IT CONTAINS ONE NODE FROM EACH OF THE
048   ;;; JUSTIFICATIONS IN THE SL AND CP SETS, SUCH THAT EACH OF THESE NODES IS RESPONSIBLE
049   ;;; FOR THE INVALIDITY OF THE CORRESPONDING JUSTIFICATION.
050
051   ;;; TMS-CONSEQUENCES
052   ;;; THIS IS A LIST OF ALL NODES WHICH MENTION THE PARTICULAR NODE IN ONE OR MORE
053   ;;; OF THEIR JUSTIFICATIONS.
054
055   ;;; TMS-EXTERNAL-NAME
056   ;;; THIS IS THE THING IN THE EXTERNAL SYSTEM TO WHICH THE NODE IS CONNECTED.  ALL
057   ;;; SIGNALLING FUNCTIONS, WHICH SIGNAL THE EXTERNAL SYSTEM OF CHANGES OF STATUS,
058   ;;; USE THIS EXTERNAL NAME AS THEIR ARGUMENTS.  THAT IS, THE EXTERNAL SYSTEM IS
059   ;;; NOTIFIED IN TERMS OF ITS OWN DATA STRUCTURES, AND NOT IN TERMS OF THE INTERNAL
060   ;;; TMS NODES.
061
062   ;;; TMS-NODE-MARK
063   ;;; THIS IS A BIT USED TO IMPROVE THE EFFICIENCY OF THE ALGORITHMS USED TO MANIPULATE
064   ;;; SETS OF NODES.
065
066   ;;; TMS-EXPLAIN-MARK
067   ;;; THIS IS A BIT USED TO IMPROVE THE EFFICIENCY OF THE EXPLANATION ALGORITHMS.  IT
068   ;;; SERVES BASICALLY THE SAME FUNCTION AS TMS-NODE-MARK, BUT IS A DIFFERENT BIT SO
069   ;;; THAT EXPLANATIONS PRODUCED DURING BREAKPOINTS, ETC. WILL NOT SCREW UP ONGOING
070   ;;; SET COMPUTATIONS.
```

```
001
002   ;;; THE FOLLOWING ATTRIBUTES OF NODES ARE KEPT ON THE PROPERTY LISTS OF NODES:
003
004   ;;; TMS-SIGNAL-RECALLING-FUNCTION
005   ;;; THIS OPTION MAY BE ATTACHED TO ANY NODE AS A FUNCTION TO BE CALLED WITH THE
006   ;;; EXTERNAL NAME OF THE NODE AS ARGUMENT WHENEVER THE NODE CHANGES STATUS FROM
007   ;;; OUT TO IN.
008   ;;; IT IS IGNORED IF IT IS THE SYMBOL 'IGNORE.
009
010   ;;; TMS-SIGNAL-FORGETTING-FUNCTION
011   ;;; THIS OPTION MAY BE ATTACHED TO ANY NODE AS A FUNCTION TO BE CALLED WITH THE
012   ;;; EXTERNAL NAME OF THE NODE AS ARGUMENT WHENEVER THE NODE CHANGES STATUS FROM
013   ;;; IN TO OUT.
014   ;;; IT IS IGNORED IF IT IS THE SYMBOL 'IGNORE.
015
016   ;;; TMS-CP-JUSTIFICATIONS
017   ;;; AS DESCRIBED ABOVE.  MOST NODES WON'T HAVE ONE, SO IT ISN'T IN THE BASIC NODE STRUCTURE.
018
019   ;;; TMS-CP-CONSEQUENT-LIST
020   ;;; THIS IS A LIST ATTACHED TO A NODE IF THE NODE IS THE "CONSEQUENT" OF ANY CONDITIONAL
021   ;;; PROOF JUSTIFICATIONS.  IF SUCH JUSTIFICATIONS EXIST, THE NODES THEY JUSTIFY ARE PUT
022   ;;; ON THIS LIST.  THIS IS TO DETERMINE THAT FINDINDEPS SHOULD BE DONE TO PRODUCE NEW
023   ;;; SUPPORT-LIST JUSTIFICATIONS FROM CONDITIONAL PROOF JUSTIFICATIONS WHEN THE CONSEQUENTS
024   ;;; OF THE CONDITIONAL PROOFS COME IN.
025
026   ;;; TMS-CONTRADICTION-NAME
027   ;;; THIS IS ATTACHED TO A NODE WHICH IS A CONTRADICTION.  IT IS SOME ARBITRARY
028   ;;; OBJECT THE EXTERNAL SYSTEM HAS CALLED THE NAME OF THE CONTRADICTION.
029
030   ;;; TMS-CONTRADICTION-TYPE
031   ;;; THIS IS ATTACHED TO A NODE WHICH IS A CONTRADICTION.  IT IS SOME ARBITRARY
032   ;;; OBJECT THE EXTERNAL SYSTEM HAS CALLED THE TYPE OF THE CONTRADICTION.
033
034   ;;; TMS-CONTRADICTION-MARK
035   ;;; THIS IS A MARK USED TO DISTINGUISH NODES DECLARED TO BE CONTRADICTIONS FROM
036   ;;; NORMAL NODES.
037
038   ;;; TMS-CONTRADICTION-FUNCTION
039   ;;; THIS IS AN OPTIONAL ATTACHMENT TO A CONTRADICTION NODE WHICH CAN POINT TO
040   ;;; A FUNCTION TO CALL WITH THE EXTERNAL NAME OF THE CONTRADICTION AS ARGUMENT
041   ;;; WHENEVER THE CONTRADICTION NODE COMES IN AND NO ASSUMPTIONS CAN BE FOUND
042   ;;; AMONG ITS SUPPORTERS - THAT IS, A CONTRADICTION WHICH CAN'T BE REMOVED.
043
044   ;;; TMS-CONTRADICTION-NOGOODS
045   ;;; THIS IS A PIECE OF DEBUGGING INFORMATION ATTACHED TO CONTRADICTIONS WHICH
046   ;;; LISTS THE NOGOODS THAT HAVE BEEN PRODUCED IN RESPONSE TO THE CONTRADICTION
047   ;;; COMING IN.
048
049   ;;; TMS-NOGOOD-ASSUMPTIONS
050   ;;; THIS IS A PIECE OF DEBUGGING INFORMATION ATTACHED TO NOGOODS WHICH LIST
051   ;;; THE ASSUMPTIONS FOUND BY THE BACKTRACKER FOR THE CONTRADICTION WHICH
052   ;;; PRODUCED THE NOGOOD.
053
054   ;;; TMS-NOGOOD-CONTRADICTION
055   ;;; THIS PIECE OF DEBUGGING INFORMATION ATTACHES THE CONTRADICTION THAT PRODUCED
056   ;;; A NOGOOD TO THAT NOGOOD.
```

```
001
002   ;;; CURRENT STORAGE ORGANIZATION OF THE SPECIAL DATA STRUCTURE:
003
004   ;;; TMS NODES ARE HUNKS, ORGANIZED AS FOLLOWS:
005   ;;; THE TWO BITS FIELDS ARE USED TO STORE THE VARIOUS SMALL MARKERS AS
006   ;;; PARTS OF FIXNUMS.  THERE ARE TWO SUCH FIELDS SO THAT THE FIXNUMS
007   ;;; INVOLVED WILL BE SMALL ENOUGH TO NOT REQUIRE NUMBER-CONSING.
008   ;;; AN ADDITIONAL NOTE:  AS WILL BE DESCRIBED LATER, NOGOODS ARE NOT NODES,
009   ;;; BUT ARE SYMBOLS WITH NODES ATTACHED.
010
011   ;;; SLOT        USE
012
013   ;;; 0   PROPERTY LIST
014   ;;; 1   TMS-EXTERNAL-NAME
015   ;;; 2   TMS-SL-JUSTIFICATIONS
016   ;;; 3   TMS-SUPPORTING-JUSTIFICATION
017   ;;; 4   TMS-SUPPORTING-NODES
018   ;;; 5   TMS-CONSEQUENCES
019   ;;; 6   TMS-BITS1
020   ;;;             TMS-NODE-MARK
021   ;;;             TMS-TMP-MARK
022   ;;;             TMS-NOTED-MARK
023   ;;;             TMS-FINDINDEP-MARK
024   ;;;             TMS-SUBORDINATES-MARK
025   ;;;             TMS-EXPLAIN-MARK
026   ;;; 7   TMS-BITS2
027   ;;;             TMS-SUPPORT-STATUS (2 BITS)
028   ;;;             TMS-SUPERIORS-MARK (2 BITS)
029
030   ;;; THIS MACRO IS USED TO ALTER COMPONENTS OF DATA STRUCTURES.
031   ;;; THE FORMAT IS (MAKE (COMPONENT OBJECT) VALUE).
032
033   (DEFMAC MAKE (X Y)
034           (CONS (IMPLODE (APPEND '(T M S - M A K E -) (CDDDDR (EXPLODEC (CAR X)))))
035                 (APPEND (CDR X) (LIST Y))))
036
037   ;;; THIS MACRO PRODUCES ACCESSING FUNCTIONS FOR HUNK STRUCTURES.
038
039   (DEFMAC HUNKFN (NAME SLOT)
040           (LET ((CN (IMPLODE (APPEND '(T M S - M A K E -) (CDDDDR (EXPLODEC NAME))))))
041                `(PROGN 'COMPILE
042                        (DEFUN ,NAME (NODE)
043                               (CXR ,SLOT NODE))
044                        (DEFUN ,CN (NODE NEW)
045                               (RPLACX ,SLOT NODE NEW)))))
046
047   (HUNKFN TMS-EXTERNAL-NAME 1)
048   (HUNKFN TMS-SL-JUSTIFICATIONS 2)
049   (HUNKFN TMS-SUPPORTING-JUSTIFICATION 3)
050   (HUNKFN TMS-SUPPORTING-NODES 4)
051   (HUNKFN TMS-CONSEQUENCES 5)
052   (HUNKFN TMS-BITS1 6)
053   (HUNKFN TMS-BITS2 7)
```

```
001
002  ;;; THESE FUNCTIONS DEFINE THE VARIOUS BIT STRUCTURES USED IN NODES.
003
004  ;;; THIS MACRO PRODUCES ACCESSING FUNCTIONS FOR BIT STRUCTURES.
005
006  (DEFMAC BITFN (NAME POS)
007          (LET ((CN (IMPLODE (APPEND '(T M S - M A K E -) (CDDDDR (EXPLODEC NAME))))))
008               `(PROGN 'COMPILE
009                       (DEFUN ,NAME (NODE)
010                              (BTON (TMS-BITS1 NODE) ,POS))
011                       (DEFUN ,CN (NODE NEW)
012                              (COND (NEW (MAKE (TMS-BITS1 NODE)
013                                               (BOR (TMS-BITS1 NODE) ,POS)))
014                                    (T (MAKE (TMS-BITS1 NODE)
015                                             (BCLR (TMS-BITS1 NODE) ,POS))))))))
016
017  (BITFN TMS-NODE-MARK 1.)
018  (BITFN TMS-TMP-MARK 2.)
019  (BITFN TMS-NOTED-MARK 4.)
020  (BITFN TMS-FINDINDEP-MARK 8.)
021  (BITFN TMS-SUBORDINATES-MARK 16.)
022  (BITFN TMS-EXPLAIN-MARK 32.)
023
024  ;;; THESE DEFINE THE REMAINING MULTI-BIT FIELDS.
025
026  (DEFUN TMS-SUPPORT-STATUS (NODE)
027         (LET ((V (BAND (TMS-BITS2 NODE) 3.)))
028              (COND ((= V 1.) 'OUT)
029                    ((= V 2.) 'IN))))
030
031  (DEFUN TMS-MAKE-SUPPORT-STATUS (NODE NEW)
032          (COND ((EQUAL NEW 'IN) (MAKE (TMS-BITS2 NODE) (BOR (BCLR (TMS-BITS2 NODE) 3.) 2.)))
033                ((EQUAL NEW 'OUT) (MAKE (TMS-BITS2 NODE) (BOR (BCLR (TMS-BITS2 NODE) 3.) 1.)))
034                (T (MAKE (TMS-BITS2 NODE) (BCLR (TMS-BITS2 NODE) 3.)))))
035
036  (DEFUN TMS-SUPERIORS-MARK (NODE)
037         (LET ((V (BAND (TMS-BITS2 NODE) 12.)))
038              (COND ((= V 4.) 'NO)
039                    ((= V 8.) 'YES))))
040
041  (DEFUN TMS-MAKE-SUPERIORS-MARK (NODE NEW)
042          (COND ((EQUAL NEW 'YES) (MAKE (TMS-BITS2 NODE) (BOR (BCLR (TMS-BITS2 NODE) 12.) 8.)))
043                ((EQUAL NEW 'NO) (MAKE (TMS-BITS2 NODE) (BOR (BCLR (TMS-BITS2 NODE) 12.) 4.)))
044                (T (MAKE (TMS-BITS2 NODE) (BCLR (TMS-BITS2 NODE) 12.)))))
```

```
001
002  ;;; THIS MACRO IS USED TO DEFINE ACCESSING FUNCTIONS FOR PROPERTY-LIST STRUCTURES.
003
004  (DEFMAC ACCESSFN (SLOTN PROPN)
005          (LET ((CNAME (IMPLODE (APPEND '(T M S - M A K E -) (CDDDDR (EXPLODEC SLOTN)))))
006                (PN (OR PROPN SLOTN)))
007               `(PROGN 'COMPILE
008                       (DEFUN ,SLOTN (NODE) (GET NODE ',PN))
009                       (DEFUN ,CNAME (NODE NEW)
010                              (COND (NEW (PUTPROP NODE NEW ',PN))
011                                    (T (REMPROP NODE ',PN))))))))
012
013  (ACCESSFN TMS-SIGNAL-RECALLING-FUNCTION)
014  (ACCESSFN TMS-SIGNAL-FORGETTING-FUNCTION)
015  (ACCESSFN TMS-CP-JUSTIFICATIONS)
016  (ACCESSFN TMS-CP-CONSEQUENT-LIST)
017  (ACCESSFN TMS-CONTRADICTION-NAME)
018  (ACCESSFN TMS-CONTRADICTION-TYPE)
019  (ACCESSFN TMS-CONTRADICTION-MARK)
020  (ACCESSFN TMS-CONTRADICTION-FUNCTION)
021  (ACCESSFN TMS-CONTRADICTION-NOGOODS)
022  (ACCESSFN TMS-NOGOOD-ASSUMPTIONS)
023  (ACCESSFN TMS-NOGOOD-CONTRADICTION)
024
025  ;;; THESE FUNCTIONS INSTALL SIGNALLING FUNCTIONS AND THEM CALL THEM IF REQUIRED.
026
027  (DEFUN TMS-INSTALL-SIGNAL-FORGETTING-FUNCTION (NODE FUN)
028         (MAKE (TMS-SIGNAL-FORGETTING-FUNCTION NODE) FUN)
029         (AND FUN (TMS-IS-OUT NODE) (FUNCALL FUN (TMS-EXTERNAL-NAME NODE))))
030
031  (DEFUN TMS-INSTALL-SIGNAL-RECALLING-FUNCTION (NODE FUN)
032         (MAKE (TMS-SIGNAL-RECALLING-FUNCTION NODE) FUN)
033         (AND FUN (TMS-IS-IN NODE) (FUNCALL FUN (TMS-EXTERNAL-NAME NODE))))
034
035  ;;; THIS FUNCTION GENERATES A VIRGIN NODE WITH A GIVEN EXTERNAL NAME.
036  ;;; SINCE IT HAS NO JUSTIFICATIONS, IT IS OUT.
037
038  (DEFUN TMS-MAKE-NODE (NAME)
039         (LET ((NODE (MAKHUNK 8.)))
040              (MAKE (TMS-SUPPORT-STATUS NODE) 'OUT)
041              (MAKE (TMS-EXTERNAL-NAME NODE) NAME)
042              NODE))
```

```
001
002   ;;; THE FOLLOWING DEFINE THE STRUCTURE OF JUSTIFICATIONS.
003   ;;; JUSTIFICATIONS ARE ALWAYS PAIRS:  THE FIRST PART OF WHICH IS THE INTERNAL
004   ;;; TMS DATA STRUCTURE, AND THE SECOND PART OF WHICH IS THE EXTERNAL REPRESENTATION
005   ;;; OF THE JUSTIFICATION.
006   ;;; THE DIFFERENT TYPES OF WAYS A SUPPORTING NODE CAN AFFECT A JUSTIFICATION ARE
007   ;;; EXPLICIT IN THE STRUCTURE OF THE JUSTIFICATION.  THUS WITHIN ANY SUBLIST
008   ;;; (THE TMS-SL-JUSTIFICATION-INLIST, FOR EXAMPLE), THE ORDER OF THE NODES
009   ;;; LISTED CANNOT MATTER.
010
011   ;;; GENERAL PAIR STRUCTURE:
012
013   (DEFUN TMS-JUSTIFICATION-ARGUMENT (JUST) (CDR JUST))
014   (DEFUN TMS-JUSTIFICATION (JUST) (CAR JUST))
015
016   ;;; SUPPORT-LIST (SL) JUSTIFICATION STRUCTURE:
017
018   (DEFUN TMS-SL-JUSTIFICATION-INLIST (JUST) (CAR (TMS-JUSTIFICATION JUST)))
019   (DEFUN TMS-SL-JUSTIFICATION-OUTLIST (JUST) (CDR (TMS-JUSTIFICATION JUST)))
020   (DEFUN TMS-MAKE-SL-JUSTIFICATION (INLIST OUTLIST EXTARG) (CONS (CONS INLIST OUTLIST) EXTARG))
021
022   ;;; CONDITIONAL-PROOF (CP) JUSTIFICATION STRUCTURE:
023
024   (DEFUN TMS-CP-JUSTIFICATION-CONSEQUENT (JUST) (CAR (TMS-JUSTIFICATION JUST)))
025   (DEFUN TMS-CP-JUSTIFICATION-IN-HYPOTHESES (JUST) (CADR (TMS-JUSTIFICATION JUST)))
026   (DEFUN TMS-CP-JUSTIFICATION-OUT-HYPOTHESES (JUST) (CDDR (TMS-JUSTIFICATION JUST)))
027   (DEFUN TMS-MAKE-CP-JUSTIFICATION (CONSEQUENT INHYPOTHESES OUTHYPOTHESES EXTARG)
028           (CONS (CONS CONSEQUENT (CONS INHYPOTHESES OUTHYPOTHESES)) EXTARG))
029
030
031   ;;; THESE FUNCTIONS PREVENT RECORDING DUPLICATE JUSTIFICATIONS FOR NODES.
032   ;;; MY THEORY OF THIS IS TRADITIONALLY FUZZY.  IN A SYSTEM LIKE THE CURRENT
033   ;;; ONE IN WHICH THERE ARE EXTERNAL FORMS FOR JUSTIFICATIONS, IT SEEMS LIKE
034   ;;; JUSTIFICATIONS WITH DISTINCT EXTERNAL FORMS SHOULD BE MAINTAINED SEPARATELY,
035   ;;; SINCE THE RETRACTION OF ONE EXTERNAL FORM MIGHT NOT MEAN THE RETRACTION OF
036   ;;; ALL INTERNALLY-IDENTICAL JUSTIFICATIONS.  IN A SINGLE-PURPOSE, INTEGRATED
037   ;;; SYSTEM SOME BETTER STRATEGY MIGHT BE POSSIBLE.
038
039   (DEFUN TMS-SL-JUSTIFICATION-MEMBER (JUST JUSTS)
040          (DO ((JS JUSTS (CDR JS)))
041              ((NULL JS))
042              (AND (EQUAL (TMS-JUSTIFICATION-ARGUMENT JUST)
043                          (TMS-JUSTIFICATION-ARGUMENT (CAR JS)))
044                   (TMS-EQUAL-LIST (TMS-SL-JUSTIFICATION-INLIST JUST)
045                                   (TMS-SL-JUSTIFICATION-INLIST (CAR JS)))
046                   (TMS-EQUAL-LIST (TMS-SL-JUSTIFICATION-OUTLIST JUST)
047                                   (TMS-SL-JUSTIFICATION-OUTLIST (CAR JS)))
048                   (RETURN T))))
049
050   ;;;CP
051   (DEFUN TMS-CP-JUSTIFICATION-MEMBER (JUST JUSTS)
052          (DO ((JS JUSTS (CDR JS)))
053              ((NULL JS))
054              (AND (EQ (TMS-CP-JUSTIFICATION-CONSEQUENT JUST)
055                       (TMS-CP-JUSTIFICATION-CONSEQUENT (CAR JS)))
056                   (EQUAL (TMS-JUSTIFICATION-ARGUMENT JUST)
057                          (TMS-JUSTIFICATION-ARGUMENT (CAR JS)))
058                   (TMS-EQUAL-LIST (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST)
059                                   (TMS-CP-JUSTIFICATION-IN-HYPOTHESES (CAR JS)))
060                   (TMS-EQUAL-LIST (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST)
061                                   (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES (CAR JS)))
062                   (RETURN T))))
063
064   ;;; THIS IS EQUAL SPECIALIZED FOR LISTS TREATING HUNKS AS ATOMS.
065
066   (DEFUN TMS-EQUAL-LIST (X Y)
067          (PROG ()
068            LP  (COND ((NULL X) (RETURN (NULL Y)))
069                      ((NULL Y) (RETURN NIL))
070                      ((EQ (CAR X) (CAR Y))
071                       (SETQ X (CDR X))
072                       (SETQ Y (CDR Y))
073                       (GO LP))
074                      (T (RETURN NIL)))))
```

```
801
802   ;;; THESE DEFINE THE FORMAT OF INTERNALLY GENERATED FACTS (LIKE NOGOODS, ETC.)
803   ;;; DUE TO THE FACT THAT CONTRADICTIONS MAY COME IN SEVERAL TIMES, THE TMS
804   ;;; MUST BE ABLE TO GENERATE INTERNAL NODES FOR NOGOODS.  SINCE THESE WILL OCCUR
805   ;;; IN EXPLANATIONS IN THE EXTERNAL SYSTEM, EITHER THE IMPLEMENTATION DETAILS
806   ;;; OF TMS NODES (AS HUNKS OR PLISTS, ETC.) MUST BE MADE AVAILABLE TO THE EXTERNAL
807   ;;; USER, OR ELSE A SIMPLE EXTERNAL STRUCTURE CAN BE GENERATED FOR THESE NODES.
808   ;;; I HAVE TAKEN THE LATTER APPROACH.  NOGOODS ARE SYMBOLS (CALLED INTERNAL FACTS).
809   ;;; THEY HAVE A SYMBOLIC ASSERTION-LIKE STATEMENT ATTACHED TO THEM, DENOTING THE
810   ;;; REASONS FOR THEIR CREATION, AS WELL AS A TMS NODE ATTACHED.
811
812   ;;; THIS FUNCTION CHECKS WHETHER AN ATOM IS AN INTERNAL TMS FACT.
813
814   (ACCESSFN TMS-FACTP)
815
816   ;;; THIS FUNCTION GETS THE STATEMENT OF AN INTERNAL TMS FACT.
817
818   (ACCESSFN TMS-FACT-STATEMENT)
819
820   ;;; THIS FUNCTION GETS THE TMS NODE OF AN INTERNAL TMS FACT.
821
822   (ACCESSFN TMS-FACT-NODE)
823
824   ;;; TMS-MAKE-FACT GENERATES A NEW INTERNAL FACT OF A GIVEN TYPE
825   ;;; (WHICH IS SPLICED INTO THE NAME OF THE FACT) AND A STATEMENT.
826
827   (DEFUN TMS-MAKE-FACT (TYPE STATEMENT)
828          (LET ((FACT (TMS-GENS TYPE)))
829               (MAKE (TMS-FACTP FACT) T)
830               (MAKE (TMS-FACT-NODE FACT) (TMS-MAKE-NODE FACT))
831               (MAKE (TMS-FACT-STATEMENT FACT) STATEMENT)
832               (MAKE (TMS-SIGNAL-RECALLING-FUNCTION FACT) 'IGNORE)
833               (MAKE (TMS-SIGNAL-FORGETTING-FUNCTION FACT) 'IGNORE)
834               FACT))
```

```
001
002  (COMMENT TRUTH MAINTENANCE SUPPORT FUNCTIONS)
003
004  ;;; THE ANTECEDENTS OF A NODE IS THE SET OF OTHER NODES IN ITS IMMEDIATE WELL-FOUNDED
005  ;;; SUPPORT.  OUT NODES HAVE NO WELL-FOUNDED SUPPORT, AND SO NO ANTECEDENTS.
006
007  (DEFUN TMS-ANTECEDENTS (NODE) (AND (TMS-IS-IN NODE) (TMS-SUPPORTING-NODES NODE)))
008
009  ;;; TMS-IS-IN AND TMS-IS-OUT TEST SINGLE NODES FOR INNESS OR OUTNESS.
010
011  (DEFUN TMS-IS-IN (NODE) (EQ (TMS-SUPPORT-STATUS NODE) 'IN))
012  (DEFUN TMS-IS-OUT (NODE) (EQ (TMS-SUPPORT-STATUS NODE) 'OUT))
013
014  ;;; TMS-ARE-IN AND TMS-ARE-OUT TEST LISTS OF NODES TO SEE IF ALL ARE IN OR OUT.
015
016  (DEFUN TMS-ARE-IN (NODES)
017         (DO ((NL NODES (CDR NL)))
018             ((NULL NL) T)
019             (OR (TMS-IS-IN (CAR NL)) (RETURN NIL))))
020
021  (DEFUN TMS-ARE-OUT (NODES)
022         (DO ((NL NODES (CDR NL)))
023             ((NULL NL) T)
024             (OR (TMS-IS-OUT (CAR NL)) (RETURN NIL))))
025
026  ;;; THIS FUNCTION RETURNS A LIST OF ALL JUSTIFICATIONS POSSESSED BY A NODE.
027
028  (DEFUN TMS-JUSTIFICATIONS (NODE)
029         (APPEND (TMS-SL-JUSTIFICATIONS NODE) (TMS-CP-JUSTIFICATIONS NODE) NIL))
030
031  ;;; THIS FUNCTION ADDS A NODE TO THE LIST OF CONSEQUENCES OF A NODE
032  ;;; IF IT WASN'T THERE ALREADY.
033
034  (DEFUN TMS-ADD-CONSEQUENCE (NODE CONSEQUENCE)
035         (OR (MEMQ CONSEQUENCE (TMS-CONSEQUENCES NODE))
036             (MAKE (TMS-CONSEQUENCES NODE)
037                   (CONS CONSEQUENCE (TMS-CONSEQUENCES NODE)))))
038
039  ;;; TMS-AFFECTED-CONSEQUENCES RETURNS A LIST OF JUST THOSE CONSEQUENCES
040  ;;; OF A NODE WHICH ACTUALLY MENTION THE NODE IN THEIR CURRENT SUPPORTING-NODES.
041  ;;; THIS IS ALL THE NODES WHICH ACTUALLY DEPEND ON THE GIVEN NODE DIRECTLY.
042
043  (DEFUN TMS-AFFECTED-CONSEQUENCES (NODE)
044         (DO ((CL (TMS-CONSEQUENCES NODE) (CDR CL))
045              (ANS NIL))
046             ((NULL CL) ANS)
047             (AND (MEMQ NODE (TMS-SUPPORTING-NODES (CAR CL)))
048                  (PUSH (CAR CL) ANS))))
049
050  ;;; TMS-AFFECTS-NODES IS BASICALLY THE PREDICATE (NOT (NULL (TMS-AFFECTED-CONSEQUENCES X))).
051  ;;; SINCE IT JUST TESTS FOR THE EXISTENCE OF AFFECTED CONSEQUENCES, IT DOESN'T HAVE TO CONS.
052
053  (DEFUN TMS-AFFECTS-NODES (NODE)
054         (DO ((CL (TMS-CONSEQUENCES NODE) (CDR CL)))
055             ((NULL CL) NIL)
056             (AND (MEMQ NODE (TMS-SUPPORTING-NODES (CAR CL))) (RETURN T))))
057
058  ;;; TMS-BELIEVED-CONSEQUENCES RETURNS A LIST OF ALL IN NODES AMONG THE AFFECTED CONSEQUENCES
059  ;;; OF A NODE.
060
061  (DEFUN TMS-BELIEVED-CONSEQUENCES (NODE)
062         (DO ((CL (TMS-CONSEQUENCES NODE) (CDR CL))
063              (ANS NIL))
064             ((NULL CL) ANS)
065             (AND (MEMQ NODE (TMS-ANTECEDENTS (CAR CL)))
066                  (PUSH (CAR CL) ANS))))
```

```
001
002   ;;; THE FOLLOWING FUNCTIONS RETURN ALL NODES OF THE SPECIFIED TYPE
003   ;;; IN RELATIONSHIP TO THE GIVEN NODE.
004
005   ;;; THIS RETURNS ALL NODES REACHED BY RECURSIVELY TAKING ANTECEDENTS.
006   ;;; TMS-MARK-FOUNDATIONS COLLECTS UP A NON-REPETITIVE LIST, AND
007   ;;; TMS-FOUNDATIONS UNMARKS AND RETURNS THE LIST.
008
009   (DEFUN TMS-FOUNDATIONS (NODE)
010          (LET ((NODELIST (MAPCAN 'TMS-MARK-FOUNDATIONS (TMS-ANTECEDENTS NODE))))
011               (MAPC '(LAMBDA (N) (MAKE (TMS-EXPLAIN-MARK N) NIL))
012                     NODELIST)))
013
014   (DEFUN TMS-MARK-FOUNDATIONS (NODE)
015          (COND ((NULL (TMS-EXPLAIN-MARK NODE))
016                 (MAKE (TMS-EXPLAIN-MARK NODE) T)
017                 (CONS NODE (MAPCAN 'TMS-MARK-FOUNDATIONS (TMS-ANTECEDENTS NODE))))))
018
019   ;;; THIS RETURNS ALL NODES REACHED BY RECURSIVELY TAKING BELIEVED CONSEQUENCES.
020   ;;; TMS-MARK-REPERCUSSIONS COLLECTS UP A NON-REPETITIVE LIST, AND
021   ;;; TMS-ALL-REPERCUSSIONS UNMARKS AND RETURNS THE LIST.
022
023   (DEFUN TMS-REPERCUSSIONS (NODE)
024          (LET ((NODELIST (MAPCAN 'TMS-MARK-REPERCUSSIONS (TMS-BELIEVED-CONSEQUENCES NODE))))
025               (MAPC '(LAMBDA (N) (MAKE (TMS-EXPLAIN-MARK N) NIL))
026                     NODELIST)))
027
028   (DEFUN TMS-MARK-REPERCUSSIONS (NODE)
029          (COND ((NULL (TMS-EXPLAIN-MARK NODE))
030                 (MAKE (TMS-EXPLAIN-MARK NODE) T)
031                 (CONS NODE (MAPCAN 'TMS-MARK-REPERCUSSIONS (TMS-BELIEVED-CONSEQUENCES NODE))))))
032
033   ;;; TMS-PREMISES AND ITS SUBFUNCTIONS COLLECT UP A LIST OF ALL PREMISES IN
034   ;;; THE RECURSIVE WELL-FOUNDED SUPPORT OF A NODE.  PREMISES ARE NODES WHICH
035   ;;; ARE IN BUT DEPEND ON NO NODES (THAT IS, HAVE NO ANTECEDENTS.)
036
037   (DEFUN TMS-PREMISES (NODE)
038          (LET ((PL (TMS-PREMISES1 NODE)))
039               (TMS-PREMISES2 NODE)
040               PL))
041
042   (DEFUN TMS-PREMISES1 (NODE)
043          (COND ((NOT (TMS-EXPLAIN-MARK NODE))
044                 (MAKE (TMS-EXPLAIN-MARK NODE) T)
045                 (COND ((TMS-ANTECEDENTS NODE) (MAPCAN 'TMS-PREMISES1 (TMS-ANTECEDENTS NODE)))
046                       (T (AND (TMS-IS-IN NODE) (LIST NODE)))))))
047
048   (DEFUN TMS-PREMISES2 (NODE)
049          (COND ((TMS-EXPLAIN-MARK NODE)
050                 (MAKE (TMS-EXPLAIN-MARK NODE) NIL)
051                 (MAPC 'TMS-PREMISES2 (TMS-ANTECEDENTS NODE)))))
052
053   ;;; TMS-ASSUMPTIONS AND ITS SUBFUNCTIONS COLLECT UP ALL ASSUMPTIONS INVOLVED
054   ;;; IN THE WELL-FOUNDED SUPPORT OF A NODE.  ASSUMPTIONS ARE NODES WHICH ARE IN AND DEPEND
055   ;;; UPON OTHER NODES WHICH ARE OUT, THAT IS, DEPEND UPON INCOMPLETE INFORMATION.
056
057   (DEFUN TMS-ASSUMPTIONS (NODE)
058          (LET ((AL (TMS-ASSUMPTIONS1 NODE)))
059               (TMS-ASSUMPTIONS2 NODE)
060               AL))
061
062   (DEFUN TMS-ASSUMPTIONS1 (NODE)
063          (COND ((NOT (TMS-EXPLAIN-MARK NODE))
064                 (MAKE (TMS-EXPLAIN-MARK NODE) T)
065                 (PROG (FLAG ANS)
066                       (MAPCAN '(LAMBDA (A)
067                                        (COND ((TMS-IS-OUT A) (SETQ FLAG T))
068                                              (T (SETQ ANS (NCONC (TMS-ASSUMPTIONS1 A) ANS)))))
069                               (TMS-ANTECEDENTS NODE))
070                       (RETURN (COND (FLAG (CONS NODE ANS))
071                                     (T ANS)))))))
072
073   (DEFUN TMS-ASSUMPTIONS2 (NODE)
074          (COND ((TMS-EXPLAIN-MARK NODE)
075                 (MAKE (TMS-EXPLAIN-MARK NODE) NIL)
076                 (MAPC 'TMS-ASSUMPTIONS2 (TMS-ANTECEDENTS NODE)))))
```

```
001
002   ;;; THIS KLUDGE TAKES A LIST OF NODES AND CLOBBERS IT TO BE A NON-REPETITIVE LIST
003   ;;; OF THE SAME NODES.  THE SPECIAL CASE OF TWO OR FEWER NODES IN THE LIST IS HANDLED
004   ;;; SPECIALLY FOR SPEED.
005
006   (DEFUN TMS-NODE-SET-CONDENSE (L)
007          (COND ((NULL L) NIL)
008                ((CDDR L)
009                 (PROG (PTR NEXT)
010                       (SETQ PTR L)
011                       (SETQ NEXT (CDR L))
012                       (MAKE (TMS-NODE-MARK (CAR PTR)) T)
013                       LP  (COND ((NULL NEXT)
014                                  (MAPC '(LAMBDA (E) (MAKE (TMS-NODE-MARK E) NIL)) L)
015                                  (RETURN L))
016                                 ((TMS-NODE-MARK (CAR NEXT))
017                                  (LET ((A (CDR NEXT)))
018                                       (RPLACD PTR A)
019                                       (SETQ NEXT A))
020                                  (GO LP))
021                                 (T (SETQ PTR NEXT)
022                                    (SETQ NEXT (CDR NEXT))
023                                    (MAKE (TMS-NODE-MARK (CAR PTR)) T)
024                                    (GO LP)))))
025                ((EQ (CAR L) (CADR L)) (CDR L))
026                (T L)))
027
028   ;;; THIS FUNCTION PRINTS A MESSAGE AND NODE FOR WALLPAPER & DEBUGGING PURPOSES.
029
030   (DEFUN TMS-PRINT (TEXT NODE)
031          (TERPRI) (PRINC TEXT) (PRINC '| |) (PRIN1 (TMS-EXTERNAL-NAME NODE)))
```

```
001
002  (COMMENT TRUTH MAINTENANCE SYSTEM COMMANDS)
003
004  ;;; TMS-SL-JUSTIFY CAN BE USED TO SUPPLY A NEW JUSTIFICATION FOR
005  ;;; A NODE, AND TO THEN DETERMINE ITS SUPPORT STATUS. IF THE NODE
006  ;;; LACKS WELL FOUNDED SUPPORT, TRUTH MAINTENANCE OCCURS.
007
008  ;;; THERE IS AN EXTRA LAYER OF HAIR INVOLVED IN ALL THE MAJOR EXTERNAL FUNCTIONS.
009  ;;; THE BASIC PROBLEM IS THAT SUPERFLUOUS STATUS CHANGE SIGNALLING SHOULD BE
010  ;;; AVOIDED.  SINCE ONE INVOCATION OF TRUTH MAINTENANCE CAN TRIGGER OTHER INVOCATIONS,
011  ;;; DUE TO CONTRADICTIONS AND CONDITIONAL PROOFS, THIS MEANS THAT ALL STATUS CHANGE
012  ;;; SIGNALLING SHOULD BE DELAYED UNTIL THE ACTUAL RETURN TO THE EXTERNAL SYSTEM.
013  ;;; THIS REQUIRES THAT THERE BE INTERNAL VERSIONS OF THE FUNCTIONS FOR JUSTIFYING NODES,
014  ;;; INVOKING BACKTRACKING, ETC.  THE EXTERNAL VERSIONS ALL CONSIST OF A RATHER
015  ;;; STANDARD BLOCK OF CODE WRAPPED AROUND THE CALL TO THE INTERNAL VERSION.
016
017  ;;; TMS-SL-JUSTIFY RETURNS NIL IF NO CHANGE IN STATUS OCCURRED, T OTHERWISE.
018
019  (DEFUN TMS-SL-JUSTIFY (NODE INS OUTS EXTARG)
020         (LET ((*TMS-NOTED-IN-NODES* NIL)
021               (*TMS-NOTED-OUT-NODES* NIL)
022               (OLDSTATUS (TMS-SUPPORT-STATUS NODE)))
023              (TMS-SL-JUSTIFY1 NODE INS OUTS EXTARG)
024              (TMS-TMP-SCAN)
025              (TMS-SIGNAL-CHANGES)
026              (NOT (EQ OLDSTATUS (TMS-SUPPORT-STATUS NODE)))))
027
028  ;;; TMS-SL-JUSTIFY1 RETURNS T IF THE JUSTIFICATION CAUSES TRUTH MAINTENANCE, NIL OTHERWISE.
029
030  (DEFUN TMS-SL-JUSTIFY1 (NODE INS OUTS EXTARG)
031         (LET ((JUST (TMS-MAKE-SL-JUSTIFICATION INS OUTS EXTARG)))
032              (COND ((LET ((JS (TMS-SL-JUSTIFICATIONS NODE)))
033                          (COND ((NOT (TMS-SL-JUSTIFICATION-MEMBER JUST JS))
034                                 (MAKE (TMS-SL-JUSTIFICATIONS NODE) (NCONC JS (LIST JUST)))
035                                 T)))
036                     (MAPC '(LAMBDA (N) (TMS-ADD-CONSEQUENCE N NODE)) INS)
037                     (MAPC '(LAMBDA (N) (TMS-ADD-CONSEQUENCE N NODE)) OUTS)
038                     (AND *TMS-SEE-JUSTIFICATIONS-SW* (TMS-PRINT '|JUSTIFYING| NODE))
039                     (EQCASE (TMS-SUPPORT-STATUS NODE)
040                             (IN NIL)
041                             (OUT (EQCASE (TMS-WF-EVAL-SL-JUSTIFICATION JUST)
042                                          (YES (TMS-TMP (LIST NODE)) T)
043                                          (NO (TMS-INSTALL-WF-SUPPORT NODE) NIL)))))))
```

```
001
002   ;;;CP
003   ;;; TMS-CP-JUSTIFY CAN BE USED TO PROVIDE A CONDITIONAL-PROOF JUSTIFICATION
004   ;;; FOR A NODE.  THE CONDITIONAL PROOF IS OF THE FORM "THE SUPPORT
005   ;;; OF CONSEQUENT RELATIVE TO THE HYPOTHESES."  IT IS EQUIVALENT TO A SUPPORT
006   ;;; LIST JUSTIFICATION CONTAINING THE OTHER NODES SUPPORTING SUCH A PROOF.
007
008   ;;; TMS-CP-JUSTIFY RETURNS NIL IF NO CHANGE IN STATUS OCCURRED, T OTHERWISE.
009
010   (DEFUN TMS-CP-JUSTIFY (NODE CONSEQUENT INHYPOTHESES OUTHYPOTHESES EXTARG)
011          (LET ((*TMS-NOTED-IN-NODES* NIL)
012                (*TMS-NOTED-OUT-NODES* NIL)
013                (OLDSTATUS (TMS-SUPPORT-STATUS NODE)))
014               (TMS-CP-JUSTIFY1 NODE CONSEQUENT INHYPOTHESES OUTHYPOTHESES EXTARG)
015               (TMS-TMP-SCAN)
016               (TMS-SIGNAL-CHANGES)
017               (NOT (EQ OLDSTATUS (TMS-SUPPORT-STATUS NODE)))))
018
019   (DEFUN TMS-CP-JUSTIFY1 (NODE CONSEQUENT INHYPOTHESES OUTHYPOTHESES EXTARG)
020          (LET ((JUST (TMS-MAKE-CP-JUSTIFICATION CONSEQUENT INHYPOTHESES OUTHYPOTHESES EXTARG)))
021               (COND ((LET ((JS (TMS-CP-JUSTIFICATIONS NODE)))
022                           (COND ((NOT (TMS-CP-JUSTIFICATION-MEMBER JUST JS))
023                                  (MAKE (TMS-CP-JUSTIFICATIONS NODE) (NCONC JS (LIST JUST)))
024                                  T)))
025                      (MAKE (TMS-CP-CONSEQUENT-LIST CONSEQUENT)
026                            (CONS NODE (TMS-CP-CONSEQUENT-LIST CONSEQUENT)))
027                      (TMS-ADD-CONSEQUENCE CONSEQUENT NODE)
028                      (MAPC '(LAMBDA (N) (TMS-ADD-CONSEQUENCE N NODE)) INHYPOTHESES)
029                      (MAPC '(LAMBDA (N) (TMS-ADD-CONSEQUENCE N NODE)) OUTHYPOTHESES)
030                      (AND *TMS-SEE-JUSTIFICATIONS-SW* (TMS-PRINT '|JUSTIFYING| NODE))
031                      (AND (EQ (TMS-SUPPORT-STATUS NODE) 'OUT)
032                           (EQCASE (TMS-WF-EVAL-CP-JUSTIFICATION JUST)
033                                   (YES
034                                    (LET ((SUPPORT
035                                           (TMS-FINDINDEP
036                                            (TMS-CP-JUSTIFICATION-CONSEQUENT JUST)
037                                            (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST)
038                                            (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST))))
039                                         (TMS-SL-JUSTIFY1 NODE (CAR SUPPORT) (CDR SUPPORT)
040                                                          (TMS-JUSTIFICATION-ARGUMENT JUST))))
041                                   (NO (TMS-INSTALL-WF-SUPPORT NODE)))))))
```

```
001
002   ;;; TMS-RETRACT REMOVES A PREMISE JUSTIFICATION FROM A NODE.
003   ;;; ACTUALLY, IT IS NOT QUITE RIGHT AT PRESENT, SINCE A RETRACTION
004   ;;; SHOULD SPECIFY THE EXACT EXTERNAL FORM OF THE PREMISE JUSTIFICATION
005   ;;; TO BE RETRACTED, LEAVING OTHER PREMISE JUSTIFICATIONS UNTOUCHED.
006   ;;; THIS WOULD OBVIATE THE HAIR PRESENTLY EXISTING WHICH CHECKS TO AVOID
007   ;;; REMOVING PREMISE JUSTIFICATIONS DERIVED FROM CONDITIONAL PROOF JUSTIFICATIONS.
008
009   ;;; PROBABLY THIS MEANS THAT TMS-RETRACT SHOULD BE GENERALIZED TO REMOVE
010   ;;; ANY TYPE OF JUSTIFICATION, GIVEN THE EXTERNAL FORM OF THE JUSTIFICATION
011   ;;; AS ARGUMENT.
012
013   (DEFUN TMS-RETRACT (NODE)
014          (LET ((*TMS-NOTED-IN-NODES* NIL)
015                (*TMS-NOTED-OUT-NODES* NIL)
016                (OLDSTATUS (TMS-SUPPORT-STATUS NODE)))
017               (TMS-RETRACT1 (LIST NODE))
018               (TMS-TMP-SCAN)
019               (TMS-SIGNAL-CHANGES)
020               (NOT (EQ OLDSTATUS (TMS-SUPPORT-STATUS NODE)))))
021
022   ;;;CP
023   (DEFUN TMS-RETRACT1 (NODELIST)
024          (DO ((NL NODELIST (CDR NL))
025               (TL NIL))
026              ((NULL NL) (AND TL (TMS-TMP TL)))
027              (DO ((JS (TMS-SL-JUSTIFICATIONS (CAR NL)) (CDR JS))
028                   (CJS (TMS-CP-JUSTIFICATIONS (CAR NL))))
029                  ((NULL JS))
030                  (LET ((JUST (CAR JS)))
031                       (COND ((AND (NULL (TMS-SL-JUSTIFICATION-INLIST JUST))
032                                   (NULL (TMS-SL-JUSTIFICATION-OUTLIST JUST))
033                                   (NOT (DO ((CS CJS (CDR CS)))
034                                            ((NULL CS))
035                                            (AND (EQ (TMS-JUSTIFICATION-ARGUMENT JUST)
036                                                     (TMS-JUSTIFICATION-ARGUMENT (CAR CS)))
037                                                 (RETURN T)))))
038                              (AND (EQ JUST (TMS-SUPPORTING-JUSTIFICATION (CAR NL)))
039                                   (PUSH (CAR NL) TL))
040                              (MAKE (TMS-SL-JUSTIFICATIONS (CAR NL))
041                                    (DELQ JUST (TMS-SL-JUSTIFICATIONS (CAR NL))))))))))
```

```
001
002   (COMMENT TRUTH MAINTENANCE PROCESSING FUNCTIONS)
003   ;;; THE TRUTH MAINTENANCE PROCESSOR:
004   ;;; TRUTH MAINTENANCE PROCESSING OCCURS WHEN THE SUPPORT STATUS
005   ;;; OF A NODE IS CHANGED.  THE MAINTENANCE PROCESSING IS
006   ;;; INITIATED BY CALLING TMS-TMP WITH THE LIST OF NODES IN
007   ;;; QUESTION.
008
009   ;;; THERE ARE TWO PHASES TO THE TRUTH MAINTENANCE PROCESS.
010   ;;; THE FIRST PHASE CONSISTS OF LOOKING FOR WELL-FOUNDED SUPPORT FOR
011   ;;; ALL NODES INVOLVED IN TRUTH MAINTENANCE.  SOME NODES MAY BE LEFT
012   ;;; WITH THEIR STATUS STILL NOT DETERMINED AT THE END OF THIS PHASE.
013   ;;; THE SECOND PHASE IS A RELAXATION PROCESS IN WHICH "NOT-WELL-FOUNDED" (NWF)
014   ;;; SUPPORT IS DERIVED FOR ALL REMAINING NODES.  THIS INVOLVES CHECKING
015   ;;; FOR SUPPORT UNDER THE ASSUMPTION THAT ALL NODES WITHOUT WELL-FOUNDED
016   ;;; SUPPORT ARE OUT.  THIS MAY BE DISCOVERED TO BE IN ERROR BY SUBSEQUENTLY
017   ;;; DERIVING SUPPORT FOR A NODE ASSUMED TO BE OUT, CAUSING FURTHER TRUTH
018   ;;; MAINTENANCE UNTIL THE DATA BASE RELAXES TO A STABLE STATE.
019
020   (DEFUN TMS-TMP (NODELIST)
021          (COND (*TMS-SEE-TMP-SW*
022                 (TERPRI)
023                 (PRINC '|TRUTH MAINTENANCE PROCESSING INITIATED|)
024                 (COND (*TMS-SEE-TMP-INVOKER-SW*
025                        (COND ((NULL (CDR NODELIST))       ;;; JUST ONE INVOKER
026                               (PRINC '| BY |)
027                               (PRIN1 (TMS-EXTERNAL-NAME (CAR NODELIST)))))))
028                 (PRINC '|.|)))
029          (SETQ *TMS-PROCESS-QUEUE* NIL)
030          (LET ((NOTED-NODES (MAPCAN 'TMS-MARK-AFFECTED-CONSEQUENCES NODELIST)))
031               (DO ((N (TMS-DEQUEUE) (TMS-DEQUEUE)))       ;;; FIRST GROVEL FOR SURE STUFF
032                   ((NULL N))
033                   (AND (NULL (TMS-SUPPORT-STATUS N)) (TMS-WF-EXAMINE N)))
034               (DO ((NL NOTED-NODES (CDR NL)))             ;;; FIND LINGERERS
035                   ((NULL NL))
036                   (OR (TMS-SUPPORT-STATUS (CAR NL)) (TMS-QUEUE (CAR NL))))
037               (DO ((N (TMS-DEQUEUE) (TMS-DEQUEUE)))       ;;; THEN GROVEL DOUBTFUL STUFF
038                   ((NULL N))
039                   (AND (NULL (TMS-SUPPORT-STATUS N)) (TMS-NWF-EXAMINE N)))
040               (MAPC '(LAMBDA (N)                          ;;; CHECK FOR BUGS IN TMS
041                        (COND ((NULL (TMS-SUPPORT-STATUS N))
042                               (PRINT (TMS-EXTERNAL-NAME N))
043                               (BREAK | NULL TMS ERROR |))))
044                     NOTED-NODES)
045               (COND (*TMS-SEE-TMP-SW*
046                      (LET ((BASE 10.) (*NOPOINT T)) (PRINT (LENGTH NOTED-NODES)))
047                      (PRINC '|NODES EXAMINED.|)))))
```

```
001
002   ;;; THIS UGLY LOOP SCANS THE LIST OF NODES INVOLVED TO SEE IF ANY ARE NOW ACTIVE
003   ;;; CONTRADICTIONS, OR ARE CP CONSEQUENCES WHICH CAN BE USED TO DERIVE NEW FINDINDEP'ED
004   ;;; SL JUSTIFICATIONS.  IF ANY ARE FOUND, TMS PROCESSING MAY OCCUR, IN WHICH CASE
005   ;;; THE SCAN MUST BE RESTARTED.
006
007   ;;;CP
008   (DEFUN TMS-TMP-SCAN ()
009          (PROG ()
010                LOOP                                          ;;; SIGH...
011                ;;; THIS CLAUSE CHECKS FOR CONTRADICTIONS AMONG NOTED NODES.
012                (AND (DO ((NL *TMS-NOTED-IN-NODES* (CDR NL)))
013                         ((NULL NL))
014                         (LET ((N (CAR NL)))
015                              (COND ((AND (TMS-IS-IN N)     ;;; THIS TEST IS NOT REDUNDANT!!!
016                                                            ;;; *TMS-NOTED-IN-NODES* CONTAINS NODES
017                                                            ;;; WHICH WERE IN UPON ENTRY TO THE
018                                                            ;;; TMS. THEY MAY NOT BE IN NOW.
019                                          (TMS-CONTRADICTION-MARK N)
020                                          (EQ (TMS-PROCESS-CONTRADICTION1
021                                               (TMS-CONTRADICTION-NAME N) N
022                                               (TMS-CONTRADICTION-TYPE N) NIL)
023                                              'FOUND-A-CULPRIT))
024                                     (RETURN T)))))
025                     (GO LOOP))
026                ;;; THIS CLAUSE CHECKS FOR CONTRADICTIONS AMONG NOTED NODES.
027                (AND (DO ((NL *TMS-NOTED-OUT-NODES* (CDR NL)))
028                         ((NULL NL))
029                         (LET ((N (CAR NL)))
030                              (COND ((AND (TMS-IS-IN N)
031                                          (TMS-CONTRADICTION-MARK N)
032                                          (EQ (TMS-PROCESS-CONTRADICTION1
033                                               (TMS-CONTRADICTION-NAME N) N
034                                               (TMS-CONTRADICTION-TYPE N) NIL)
035                                              'FOUND-A-CULPRIT))
036                                     (RETURN T)))))
037                     (GO LOOP))
038                ;;; THIS CLAUSE CHECKS FOR THE OPPORTUNITY TO MAKE NEW SUPPORT-LIST
039                ;;; JUSTIFICATIONS FROM NEWLY INNED CONSEQUENTS OF CONDITIONAL PROOFS.
040                (AND (DO ((NL *TMS-NOTED-IN-NODES* (CDR NL)))
041                         ((NULL NL))
042                         (LET ((N (CAR NL)))
043                              (AND (TMS-IS-IN N)
044                                   (TMS-CHECK-CP-CONSEQUENCES N)
045                                   (RETURN T))))
046                     (GO LOOP))
047                ;;; THIS CLAUSE CHECKS FOR THE OPPORTUNITY TO MAKE NEW SUPPORT-LIST
048                ;;; JUSTIFICATIONS FROM NEWLY INNED CONSEQUENTS OF CONDITIONAL PROOFS.
049                (AND (DO ((NL *TMS-NOTED-OUT-NODES* (CDR NL)))
050                         ((NULL NL))
051                         (LET ((N (CAR NL)))
052                              (AND (TMS-IS-IN N)
053                                   (TMS-CHECK-CP-CONSEQUENCES N)
054                                   (RETURN T))))
055                     (GO LOOP))))
```

```
001
002   ;;; TMS-CHECK-CP-CONSEQUENCES REDERIVES SUPPORT FOR CONDITIONALLY PROVEN NODES
003   ;;; WHENEVER THE CONSEQUENT OF ONE OF THEIR CONDITIONAL-PROOF JUSTIFICATIONS
004   ;;; COMES IN.
005
006   ;;;CP
007   (DEFUN TMS-CHECK-CP-CONSEQUENCES (NODE)
008          (PROG (CHANGED)
009                (MAPC '(LAMBDA (CPN)
010                         (MAPC '(LAMBDA (JUST)
011                                  (COND ((AND (EQ NODE (TMS-CP-JUSTIFICATION-CONSEQUENT JUST))
012                                              (TMS-WF-IN (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST))
013                                              (TMS-WF-OUT (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST)))
014                                         (LET ((SUPPORT
015                                                (TMS-FINDINDEP
016                                                 NODE
017                                                 (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST)
018                                                 (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST))))
019                                           (AND (TMS-SL-JUSTIFY1 CPN
020                                                                 (CAR SUPPORT)
021                                                                 (CDR SUPPORT)
022                                                                 (TMS-JUSTIFICATION-ARGUMENT JUST))
023                                                                 (SETQ CHANGED T))))))
024                                        (TMS-CP-JUSTIFICATIONS CPN)))
025                              (TMS-CP-CONSEQUENT-LIST NODE))
026                      CHANGED))
027
028   ;;; THIS FUNCTION CHECKS THE NODES INVOLVED IN TRUTH MAINTENANCE TO SEE IF ANY
029   ;;; HAVE CHANGED IN STATUS AND SHOULD BE SIGNALLED.
030   ;;; IT IS IMPORTANT THAT TMS-SIGNAL-STATUS-CHANGE DOES NOT CAUSE
031   ;;; FURTHER TRUTH MAINTENANCE UNTIL THE FOLLOWING LOOP IS COMPLETED.
032
033   (DEFUN TMS-SIGNAL-CHANGES ()
034          (MAPC '(LAMBDA (N)
035                   (TMS-SIGNAL-STATUS-CHANGE N 'IN (TMS-SUPPORT-STATUS N))
036                   (MAKE (TMS-NOTED-MARK N) NIL))
037                *TMS-NOTED-IN-NODES*)
038          (SETQ *TMS-NOTED-IN-NODES* NIL)
039          (MAPC '(LAMBDA (N)
040                   (TMS-SIGNAL-STATUS-CHANGE N 'OUT (TMS-SUPPORT-STATUS N))
041                   (MAKE (TMS-NOTED-MARK N) NIL))
042                *TMS-NOTED-OUT-NODES*)
043          (SETQ *TMS-NOTED-OUT-NODES* NIL))
044
045   ;;; STIMULATE AND DESTIMULATE SHOULD BE SUPPLIED BY THE USER AS THE
046   ;;; DEFAULT SIGNAL-RECALLING AND SIGNAL-FORGETTING FUNCTIONS.
047   ;;; THE SIGNAL-RECALLING AND SIGNAL-FORGETTING FUNCTIONS CAN ALSO BE
048   ;;; SET INDIVIDUALLY.  THESE MAY ALSO BE THE ATOM 'IGNORE, IN WHICH
049   ;;; CASE THE CHANGE WILL BE IGNORED.
050
051   (DEFUN TMS-SIGNAL-STATUS-CHANGE (NODE OLDSTATUS NEWSTATUS)
052          (COND ((EQ OLDSTATUS NEWSTATUS))
053                ((EQ NEWSTATUS 'IN)
054                 (LET ((RF (TMS-SIGNAL-RECALLING-FUNCTION NODE)))
055                      (COND ((NULL RF) (STIMULATE (TMS-EXTERNAL-NAME NODE)))
056                            ((EQ RF 'IGNORE))
057                            (T (FUNCALL RF (TMS-EXTERNAL-NAME NODE))))))
058                ((EQ NEWSTATUS 'OUT)
059                 (LET ((FF (TMS-SIGNAL-FORGETTING-FUNCTION NODE)))
060                      (COND ((NULL FF) (DESTIMULATE (TMS-EXTERNAL-NAME NODE)))
061                            ((EQ FF 'IGNORE))
062                            (T (FUNCALL FF (TMS-EXTERNAL-NAME NODE))))))
063                (T (ERROR 'TMS-SIGNAL-STATUS-CHANGE NODE 'WRNG-TYPE-ARG))))
```

```
001
002   ;;; THE FOLLOWING FUNCTIONS PERFORM THE PROCESS QUEUE MAINTENANCE OPERATIONS.
003   ;;; THE POSSIBLE CONDITIONS OF A NODE ARE AS FOLLOWS:
004   ;;; IF THE TMS-NOTED-MARK IS NIL, THEN THE NODE HAS NOT BEEN EXAMINED BY THE TMS
005   ;;; IF THE TMS-STATUS IS NON-NIL, THEN THE NODE IS QUEUED FOR TMS PROCESSING.
006
007   (DEFUN TMS-QUEUE (NODE)
008          (COND ((NOT (TMS-TMP-MARK NODE))
009                 (OR (TMS-NOTED-MARK NODE)
010                     (ERROR '|NON-NOTED NODE IN TMS-QUEUE| NODE 'WRNG-TYPE-ARG))
011                 (MAKE (TMS-TMP-MARK NODE) T)
012                 (PUSH NODE *TMS-PROCESS-QUEUE*))))
013
014   (DEFUN TMS-DEQUEUE ()
015          (LET ((NODE (POP *TMS-PROCESS-QUEUE*)))
016               (COND (NODE
017                      (MAKE (TMS-TMP-MARK NODE) NIL)
018                      NODE))))
019
020   ;;; THIS FUNCTION MARKS AND QUEUES ALL NODES WHICH MIGHT BE
021   ;;; AFFECTED BY THE CHANGE OF SUPPORT STATUS OF THE ARGUMENT NODE.
022   ;;; THE STATUS OF THE NODE BEFORE THE TMS PROCESSING IS ALSO NOTED
023   ;;; TO ALLOW NOTIFICATION OF STATUS CHANGES AT THE CONCLUSION OF TMS PROCESSING.
024
025   (DEFUN TMS-MARK-AFFECTED-CONSEQUENCES (NODE)
026          (COND ((NOT (TMS-TMP-MARK NODE))
027                 (COND ((NULL (TMS-NOTED-MARK NODE))
028                        (MAKE (TMS-NOTED-MARK NODE) T)
029                        (EQCASE (TMS-SUPPORT-STATUS NODE)
030                                (IN (PUSH NODE *TMS-NOTED-IN-NODES*))
031                                (OUT (PUSH NODE *TMS-NOTED-OUT-NODES*))
032                                (ELSE (ERROR '|STATUSLESS NODE IN TMS-MARK-AFFECTED-CONSEQUENCES|
033                                             NODE
034                                             'WRNG-TYPE-ARG)))))
035                 (MAKE (TMS-SUPPORT-STATUS NODE) NIL)
036                 (MAKE (TMS-SUPPORTING-JUSTIFICATION NODE) NIL)
037                 (MAKE (TMS-SUPPORTING-NODES NODE) NIL)
038                 (TMS-QUEUE NODE)
039                 (CONS NODE (MAPCAN 'TMS-MARK-AFFECTED-CONSEQUENCES
040                                    (TMS-AFFECTED-CONSEQUENCES NODE))))))
```

```
001
002  (COMMENT SUPPORT-CHECKING FUNCTIONS)
003
004  ;;; TMS-WF-EXAMINE RECURSIVELY CHECKS NODES FOR WELL-FOUNDED SUPPORT.
005  ;;; THAT IS, IF IT FINDS WELL-FOUNDED SUPPORT FOR A NODE, IT QUEUES UP THE
006  ;;; CONSEQUENCES OF THE NODE WHICH STILL ARE LACKING WELL-FOUNDED SUPPORT
007  ;;; TO SEE IF SUCH SUPPORT CAN NOW BE DERIVED.
008
009  (DEFUN TMS-WF-EXAMINE (NODE)
010         (LET ((NEWSTATUS (TMS-WF-STATUS NODE)))
011              (COND (NEWSTATUS
012                     (COND ((EQ NEWSTATUS 'IN) (MAKE (TMS-SUPPORT-STATUS NODE) 'IN))
013                           (T (MAKE (TMS-SUPPORT-STATUS NODE) 'OUT)))
014                     (TMS-INSTALL-WF-SUPPORT NODE)
015                     (MAPC '(LAMBDA (C) (OR (TMS-SUPPORT-STATUS C) (TMS-QUEUE C)))
016                           (TMS-CONSEQUENCES NODE)))))))
017
018  ;;; TMS-NWF-EXAMINE SELECTS DUBIOUS SUPPORT FOR A NODE.
019  ;;; IT CHECKS FOR SUPPORT UNDER THE ASSUMPTION THAT ANY NODES WITHOUT
020  ;;; WELL-FOUNDED SUPPORT MENTIONED IN JUSTIFICATIONS ARE OUT.
021  ;;; PUT ANOTHER WAY, IT EVALUATES JUSTIFICATIONS UNDER THE ASSUMPTION THAT OUT = NIL
022  ;;; IN SUPPORT-STATUSES FOR NODES.  THIS ASSUMPTION MAY NOT BE RIGHT,
023  ;;; AND THE NODE'S STATUS MAY LATER BE CHANGED IN THE RELAXATION PROCESS.
024
025  (DEFUN TMS-NWF-EXAMINE (NODE)
026         (OR (LET ((STATUS (TMS-WF-STATUS NODE)))
027                  (COND (STATUS
028                         (COND ((EQ STATUS 'IN) (MAKE (TMS-SUPPORT-STATUS NODE) 'IN))
029                               (T (MAKE (TMS-SUPPORT-STATUS NODE) 'OUT)))
030                         (TMS-INSTALL-WF-SUPPORT NODE)
031                         (TMS-NWF-PROCESS-CONSEQUENCES NODE STATUS)
032                         STATUS)))
033             (LET ((STATUS (TMS-NWF-STATUS NODE)))
034                  (EQCASE STATUS
035                          (IN (MAKE (TMS-SUPPORT-STATUS NODE) 'IN))
036                          (OUT (MAKE (TMS-SUPPORT-STATUS NODE) 'OUT)))
037                  (TMS-INSTALL-NWF-SUPPORT NODE)
038                  (TMS-NWF-PROCESS-CONSEQUENCES NODE STATUS)
039                  STATUS)))
040
041  ;;; THIS FUNCTION CHECKS THE CONSEQUENCES OF NODES FOR WHICH DUBIOUS SUPPORT
042  ;;; WAS DERIVED BY TMS-NWF-EXAMINE.  IT FIRST TRIES TO CHECK THEM FOR POSSIBLE
043  ;;; WELL-FOUNDED SUPPORT, AND FAILING THAT, SUBMITS THEM TO TMS-NWF-EXAMINE
044  ;;; TO CHECK FOR DUBIOUS SUPPORT.
045  ;;; THIS OCCURS BECAUSE IT MAY BE IMPOSSIBLE TO FIND WELL-FOUNDED SUPPORT
046  ;;; FOR A NODE EITHER BECAUSE IT IS INVOLVED IN A CIRCULARITY, OR BECAUSE
047  ;;; IT DEPENDS UPON A NODE WHICH IS INVOLVED IN A CIRCULARITY.
048
049   (DEFUN TMS-NWF-PROCESS-CONSEQUENCES (NODE STATUS)
050          (COND ((EQ STATUS 'IN)
051                 (MAPC '(LAMBDA (C)
052                          (COND ((NULL (TMS-SUPPORT-STATUS C)) (TMS-QUEUE C))
053                                ((MEMQ NODE (TMS-SUPPORTING-NODES C))
054                                 (TMS-MARK-AFFECTED-CONSEQUENCES C))))
055                       (TMS-CONSEQUENCES NODE)))
056                (T (MAPC '(LAMBDA (C)
057                            (OR (TMS-SUPPORT-STATUS C) (TMS-QUEUE C)))
058                         (TMS-CONSEQUENCES NODE)))))
```

```
001
002   ;;;TMS-WF-STATUS COMPUTES THE WELL-FOUNDED SUPPORT STATUS OF
003   ;;; A NODE FROM BOTH ITS JUSTIFICATION SETS.
004
005   (DEFUN TMS-WF-STATUS (NODE)
006          (EQCASE (TMS-WF-SL-SUPPORT NODE)
007                  (IN 'IN)
008                  (OUT (TMS-WF-CP-SUPPORT NODE))
009                  (ELSE (AND (EQ (TMS-WF-CP-SUPPORT NODE) 'IN) 'IN))))
010
011   ;;; TMS-WF-SL-SUPPORT COMPUTES THE WELL-FOUNDED SUPPORT-STATUS DERIVED
012   ;;; FROM ITS SUPPORT-LIST JUSTIFICATION SET.
013
014   (DEFUN TMS-WF-SL-SUPPORT (NODE)
015          (DO ((JS (TMS-SL-JUSTIFICATIONS NODE) (CDR JS))
016               (WF T))
017              ((NULL JS) (AND WF 'OUT))
018              (EQCASE (TMS-WF-EVAL-SL-JUSTIFICATION (CAR JS))
019                      (YES (RETURN 'IN))
020                      (NO)
021                      (ELSE (SETQ WF NIL)))))
022
023   ;;; TMS-WF-CP-SUPPORT COMPUTES THE WELL-FOUNDED SUPPORT-STATUS
024   ;;; DERIVED FROM ITS CONDITIONAL-PROOF JUSTIFICATION SET.
025
026   ;;;CP
027   (DEFUN TMS-WF-CP-SUPPORT (NODE)
028          (DO ((JS (TMS-CP-JUSTIFICATIONS NODE) (CDR JS))
029               (WF T))
030              ((NULL JS) (AND WF 'OUT))
031              (EQCASE (TMS-WF-EVAL-CP-JUSTIFICATION (CAR JS))
032                      (YES (RETURN 'IN))
033                      (NO)
034                      (ELSE (SETQ WF NIL)))))
035
036   ;;; TMS-WF-EVAL-SL-JUSTIFICATION EVALUATES AN SUPPORT-LIST JUSTIFICATION.
037   ;;; THIS MEANS CHECKING TO SEE IF THE JUSTIFICATION IS VALID.
038
039   (DEFUN TMS-WF-EVAL-SL-JUSTIFICATION (JUST)
040          (EQCASE (TMS-WF-IN (TMS-SL-JUSTIFICATION-INLIST JUST))
041                  (YES (TMS-WF-OUT (TMS-SL-JUSTIFICATION-OUTLIST JUST)))
042                  (NO 'NO)
043                  (ELSE NIL)))
044
045   ;;; TMS-WF-EVAL-CP-JUSTIFICATION EVALUATES A CONDITIONAL-PROOF JUSTIFICATION.
046
047   ;;;CP
048   (DEFUN TMS-WF-EVAL-CP-JUSTIFICATION (JUST)
049          (EQCASE (TMS-WF-IN (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST))
050                  (YES (EQCASE (TMS-WF-OUT (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST))
051                               (YES (EQCASE (TMS-SUPPORT-STATUS
052                                             (TMS-CP-JUSTIFICATION-CONSEQUENT JUST))
053                                            (IN 'YES)
054                                            (OUT 'NO)
055                                            (ELSE NIL)))
056                               (ELSE NIL)))
057                  (ELSE NIL)))
```

```
001
002   ;;; TMS-WF-IN CHECKS A LIST OF NODES FOR WELL-FOUNDED INNESS.
003
004   (DEFUN TMS-WF-IN (NODELIST)
005          (DO ((NL NODELIST (CDR NL))
006               (WF T))
007              ((NULL NL) (AND WF 'YES))
008              (EQCASE (TMS-SUPPORT-STATUS (CAR NL))
009                      (IN)
010                      (OUT (RETURN 'NO))
011                      (ELSE (SETQ WF NIL)))))
012
013   ;;; TMS-WF-OUT CHECKS A LIST OF NODES FOR WELL-FOUNDED OUTNESS.
014
015   (DEFUN TMS-WF-OUT (NODELIST)
016          (DO ((NL NODELIST (CDR NL))
017               (WF T))
018              ((NULL NL) (AND WF 'YES))
019              (EQCASE (TMS-SUPPORT-STATUS (CAR NL))
020                      (IN (RETURN 'NO))
021                      (OUT)
022                      (ELSE (SETQ WF NIL)))))
023
024   ;;; TMS-NWF-STATUS COMPUTES THE (PERHAPS UNFOUNDED) SUPPORT-STATUS OF A NODE.
025   ;;; THIS MEANS EVALUATING JUSTIFICATIONS, ETC. UNDER THE ASSUMPTION THAT
026   ;;; A SUPPORT STATUS OF NIL IS EQUIVALENT TO A SUPPORT STATUS OF OUT.
027
028   (DEFUN TMS-NWF-STATUS (NODE)
029          (EQCASE (TMS-NWF-SL-SUPPORT NODE)
030                  (IN 'IN)
031                  (OUT (EQCASE (TMS-WF-CP-SUPPORT NODE)
032                               (IN (ERROR 'TMS-NWF-STATUS NODE 'WRNG-TYPE-ARG) 'OUT)
033                               (OUT 'OUT)
034                               (ELSE 'OUT)))))
035
036   ;;; TMS-NWF-SL-SUPPORT COMPUTES THE SUPPORT-STATUS OF A NODE FROM ITS
037   ;;; SUPPORT-LIST JUSTIFICATION SET BY EQUATING 'OUT AND NIL.
038   ;;; ITS SUBFUNCTIONS ARE ANALOGOUS TO THE WELL-FOUNDED CASE FUNCTIONS ABOVE.
039
040   (DEFUN TMS-NWF-SL-SUPPORT (NODE)
041          (DO ((JS (TMS-SL-JUSTIFICATIONS NODE) (CDR JS)))
042              ((NULL JS) 'OUT)
043              (AND (TMS-NWF-EVAL-SL-JUSTIFICATION (CAR JS))
044                   (RETURN 'IN))))
045
046   (DEFUN TMS-NWF-EVAL-SL-JUSTIFICATION (JUST)
047          (AND (TMS-NWF-IN (TMS-SL-JUSTIFICATION-INLIST JUST))
048               (TMS-NWF-OUT (TMS-SL-JUSTIFICATION-OUTLIST JUST))))
049
050   (DEFUN TMS-NWF-IN (NODELIST)
051          (DO ((NL NODELIST (CDR NL)))
052              ((NULL NL) T)
053              (OR (TMS-IS-IN (CAR NL)) (RETURN NIL))))
054
055   (DEFUN TMS-NWF-OUT (NODELIST)
056          (DO ((NL NODELIST (CDR NL)))
057              ((NULL NL) T)
058              (AND (TMS-IS-IN (CAR NL)) (RETURN NIL))))
```

```
001
002  (COMMENT SUPPORT-EXTRACTION FUNCTIONS)
003
004  ;;; THIS FUNCTION IS COMPLETELY HAIRY AS IT TRIES TO FIRST CHECK TO SEE IF
005  ;;; ANY NEW SL JUSTIFICATIONS CAN BE DERIVED FROM CP JUSTIFICATIONS.
006  ;;; IT THEN INSTALLS THE FIRST VALID SL JUSTIFICATION IT CAN FIND AS THE
007  ;;; SUPPORTING-JUSTIFICATION OF THE NODE, AND EXTRACTS THE APPROPRIATE
008  ;;; SUPPORTING NODES.
009  ;;; THERE IS A CRUCIAL TIME ORDERING USED HERE.
010  ;;; RATHER THAN HAVING THE FUNCTIONS WHICH CHECK THE SET OF JUSTIFICATIONS
011  ;;; FOR A VALID JUSTIFICATION RETURN THAT JUSTIFICATION, THIS PROGRAM MAKES
012  ;;; SURE THAT THAT CHECKING OF THE JUSTIFICATION SETS IS DONE IN A PARTICULAR
013  ;;; ORDER (SL-JUSTIFICATIONS FIRST, THEN CP-JUSTIFICATIONS.)
014  ;;; THE REASON FOR THIS IS THAT THE EXTRACTION ROUTINES BELOW, WHICH COMPUTE
015  ;;; A HOPEFULLY MINIMAL SET OF NODES WHICH DETERMINE THE SUPPORT STATUS
016  ;;; OF THE CURRENT NODE (EITHER IN OR OUT).  THE DECISION OF WHICH SET TO
017  ;;; EXTRACT DEPENDS UPON WHETHER THE NODE IS IN OR OUT.
018  ;;; THUS THE STATUS CHECKING FUNCTIONS ABOVE COMPUTE NO SUPPORT SETS.
019  ;;; THIS TASK IS LEFT TO THE FOLLOWING FUNCTIONS.
020  ;;; TMS-INSTALL-WF-SUPPORT USES THE FIRST VALID JUSTIFICATION FOUND
021  ;;; IN A REPETITIVE CHECKING OF THE JUSTIFICATION SETS AS THE SUPPORTING
022  ;;; JUSTIFICATION, SINCE THIS IS THE FIRST VALID JUSTIFICATION WHICH WAS ENCOUNTERED
023  ;;; BY THE STATUS CHECKING ROUTINES.
024
025  ;;; NOTE:  THE LOGICAL NECESSITY OF THESE REPETITIVE CHECKS ALL OVER THE
026  ;;;        PLACE FOR OPPORTUNITIES TO DO FINDINDEPS SHOULD BE THOUGHT OUT
027  ;;;        CAREFULLY, AND THE CODE REORGANIZED.  ALSO, THERE IS SOME
028  ;;;        SUPERFLUOUS STUFF IN THE EXTRACTION FUNCTIONS BELOW.
029
030  ;;;CP
031  (DEFUN TMS-INSTALL-WF-SUPPORT (NODE)
032         (DO ((JS (TMS-CP-JUSTIFICATIONS NODE) (CDR JS)))
033             ((NULL JS))
034             (EQCASE (TMS-WF-EVAL-CP-JUSTIFICATION (CAR JS))
035                     (YES (LET ((SUPPORT (TMS-FINDINDEP
036                                          (TMS-CP-JUSTIFICATION-CONSEQUENT (CAR JS))
037                                          (TMS-CP-JUSTIFICATION-IN-HYPOTHESES (CAR JS))
038                                          (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES (CAR JS)))))
039                               (LET ((JUST (TMS-MAKE-SL-JUSTIFICATION
040                                            (CAR SUPPORT)
041                                            (CDR SUPPORT)
042                                            (TMS-JUSTIFICATION-ARGUMENT (CAR JS))))
043                                     (OJS (TMS-SL-JUSTIFICATIONS NODE)))
044                                    (AND (NOT (TMS-SL-JUSTIFICATION-MEMBER JUST OJS))
045                                         (MAKE (TMS-SL-JUSTIFICATIONS NODE)
046                                               (NCONC OJS (LIST JUST)))))))
047                     (ELSE)))
048         (DO ((JS (TMS-SL-JUSTIFICATIONS NODE) (CDR JS)))
049             ((NULL JS)
050              (MAKE (TMS-SUPPORTING-JUSTIFICATION NODE) NIL)
051              (MAKE (TMS-SUPPORTING-NODES NODE)
052                    (TMS-NODE-SET-CONDENSE
053                     (NCONC (MAPCAN '(LAMBDA (J) (TMS-WF-SL-JUSTIFICATION-EXTRACT J))
054                                    (TMS-SL-JUSTIFICATIONS NODE))
055                            (MAPCAN '(LAMBDA (J) (TMS-WF-CP-JUSTIFICATION-EXTRACT J))
056                                    (TMS-CP-JUSTIFICATIONS NODE))))))
057             (EQCASE (TMS-WF-EVAL-SL-JUSTIFICATION (CAR JS))
058                     (YES (MAKE (TMS-SUPPORTING-JUSTIFICATION NODE) (CAR JS))
059                          (MAKE (TMS-SUPPORTING-NODES NODE)
060                                (APPEND (TMS-SL-JUSTIFICATION-INLIST (CAR JS))
061                                        (TMS-SL-JUSTIFICATION-OUTLIST (CAR JS))
062                                        NIL))
063                          (RETURN NIL))
064                     (ELSE))))
```

```
001
002   ;;; IF THE NODE IS BEING GIVEN DUBIOUS SUPPORT BY THE NOT-WELL-FOUNDED
003   ;;; RELAXATION PROCESS, THIS FUNCTION IS USED INSTEAD OF TMS-INSTALL-WF-SUPPORT.
004
005   (DEFUN TMS-INSTALL-NWF-SUPPORT (NODE)
006          (DO ((JS (TMS-SL-JUSTIFICATIONS NODE) (CDR JS)))
007              ((NULL JS)
008               (MAKE (TMS-SUPPORTING-JUSTIFICATION NODE) NIL)
009               (MAKE (TMS-SUPPORTING-NODES NODE)
010                     (TMS-NODE-SET-CONDENSE
011                      (NCONC
012                       (MAPCAN '(LAMBDA (J) (TMS-NWF-SL-JUSTIFICATION-EXTRACT J))
013                               (TMS-SL-JUSTIFICATIONS NODE))
014                       (MAPCAN '(LAMBDA (J) (TMS-NWF-CP-JUSTIFICATION-EXTRACT J))
015                               (TMS-CP-JUSTIFICATIONS NODE))))))
016              (COND ((TMS-NWF-EVAL-SL-JUSTIFICATION (CAR JS))
017                     (MAKE (TMS-SUPPORTING-JUSTIFICATION NODE) (CAR JS))
018                     (MAKE (TMS-SUPPORTING-NODES NODE)
019                           (TMS-NODE-SET-CONDENSE (TMS-NWF-SL-JUSTIFICATION-EXTRACT (CAR JS))))
020                     (RETURN NIL)))))
021
022   ;;; THESE FUNCTIONS ARE USED TO EXTRACT A HOPEFULLY SMALL SET OF NODES AS THE
023   ;;; SET OF SUPPORTING NODES OF THE NODE.  THE EXTRACTED NODES ARE ONES THAT MUST
024   ;;; BE CHANGED TO AFFECT THE STATUS OF THE SUPPORTED NODE.
025
026   ;;; IF THE JUSTIFICATION IS VALID, THIS RETURNS THE UNION OF THE IN AND OUT SETS OF NODES
027   ;;; MENTIONED IN THE JUSTIFICATION.  IF THE JUSTIFICATION IS INVALID, IT RETURNS EITHER
028   ;;; AN OUT NODE FROM THE INLIST, OR A IN NODE FROM THE OUTLIST.
029
030   (DEFUN TMS-WF-SL-JUSTIFICATION-EXTRACT (JUST)
031          (EQCASE (TMS-WF-IN (TMS-SL-JUSTIFICATION-INLIST JUST))
032                  (YES (EQCASE (TMS-WF-OUT (TMS-SL-JUSTIFICATION-OUTLIST JUST))
033                               (YES (APPEND (TMS-SL-JUSTIFICATION-INLIST JUST)
034                                            (TMS-SL-JUSTIFICATION-OUTLIST JUST)
035                                            NIL))
036                               (NO (TMS-WF-OUT-EXTRACT (TMS-SL-JUSTIFICATION-OUTLIST JUST)))
037                               (ELSE NIL)))
038                  (NO (TMS-WF-IN-EXTRACT (TMS-SL-JUSTIFICATION-INLIST JUST)))
039                  (ELSE NIL)))
040
041   ;;; I DON'T KNOW WHAT THIS DOES ANYMORE.
042
043   ;;;CP
044   (DEFUN TMS-WF-CP-JUSTIFICATION-EXTRACT (JUST)
045          (EQCASE (TMS-WF-IN (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST))
046                  (YES (EQCASE (TMS-WF-OUT (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST))
047                               (YES (EQCASE (TMS-SUPPORT-STATUS
048                                             (TMS-CP-JUSTIFICATION-CONSEQUENT JUST))
049                                            (IN (ERROR 'TMS-WF-CP-JUSTIFICATION-EXTRACT
050                                                       JUST 'WRNG-TYPE-ARG) NIL)
051                                            (OUT (LIST (TMS-CP-JUSTIFICATION-CONSEQUENT JUST)))
052                                            (ELSE NIL)))
053                               (NO (ERROR 'TMS-WF-CP-JUSTIFICATION-EXTRACT-OUTS
054                                          JUST 'WRNG-TYPE-ARG) NIL)
055                               (ELSE NIL)))
056                  (NO (ERROR 'TMS-WF-CP-JUSTIFICATION-EXTRACT-INS JUST 'WRNG-TYPE-ARG) NIL)
057                  (ELSE NIL)))
```

```
001
002   ;;; THIS RETURNS THE ENTIRE LIST OF NODES IF ALL ARE IN,
003   ;;; OR THE FIRST OUT NODE IF THERE IS ONE.
004
005   (DEFUN TMS-WF-IN-EXTRACT (NODELIST)
006          (DO ((NL NODELIST (CDR NL))
007               (WF T))
008              ((NULL NL) (AND WF (APPEND NODELIST NIL)))
009              (EQCASE (TMS-SUPPORT-STATUS (CAR NL))
010                      (IN)
011                      (OUT (RETURN (LIST (CAR NL))))
012                      (ELSE (SETQ WF NIL)))))
013
014   ;;; THIS RETURNS THE ENTIRE LIST OF NODES IF ALL ARE OUT,
015   ;;; OR THE FIRST IN NODE IF THERE IS ONE.
016
017   (DEFUN TMS-WF-OUT-EXTRACT (NODELIST)
018          (DO ((NL NODELIST (CDR NL))
019               (WF T))
020              ((NULL NL) (AND WF (APPEND NODELIST NIL)))
021              (EQCASE (TMS-SUPPORT-STATUS (CAR NL))
022                      (IN (RETURN (LIST (CAR NL))))
023                      (OUT)
024                      (ELSE (SETQ WF NIL)))))
025
026   ;;; THESE FUNCTIONS ARE ANALOGOUS TO THE CORRESPONDING WELL-FOUNDED CASE FUNCTIONS ABOVE.
027   ;;; THE ONLY REAL DIFFERENCE IS THAT NIL IS CONSIDERED OUT IN SUPPORT STATUSES.
028
029   (DEFUN TMS-NWF-SL-JUSTIFICATION-EXTRACT (JUST)
030          (COND ((TMS-NWF-IN (TMS-SL-JUSTIFICATION-INLIST JUST))
031                 (COND ((TMS-NWF-OUT (TMS-SL-JUSTIFICATION-OUTLIST JUST))
032                        (APPEND (TMS-SL-JUSTIFICATION-INLIST JUST)
033                                (TMS-SL-JUSTIFICATION-OUTLIST JUST)
034                                NIL))
035                       (T (TMS-NWF-OUT-EXTRACT (TMS-SL-JUSTIFICATION-OUTLIST JUST)))))
036                (T (TMS-NWF-IN-EXTRACT (TMS-SL-JUSTIFICATION-INLIST JUST)))))
037
038   ;;;CP
039   (DEFUN TMS-NWF-CP-JUSTIFICATION-EXTRACT (JUST)
040          (COND ((TMS-NWF-IN (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST))
041                 (COND ((TMS-NWF-OUT (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST))
042                        (COND ((TMS-IS-IN (TMS-CP-JUSTIFICATION-CONSEQUENT JUST))
043                               (APPEND (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST)
044                                       (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST)
045                                       (LIST (TMS-CP-JUSTIFICATION-CONSEQUENT JUST))))
046                              (T (LIST (TMS-CP-JUSTIFICATION-CONSEQUENT JUST)))))
047                       (T (TMS-NWF-OUT-EXTRACT (TMS-CP-JUSTIFICATION-OUT-HYPOTHESES JUST)))))
048                (T (TMS-NWF-IN-EXTRACT (TMS-CP-JUSTIFICATION-IN-HYPOTHESES JUST)))))
049
050   (DEFUN TMS-NWF-IN-EXTRACT (NODELIST)
051          (DO ((NL NODELIST (CDR NL)))
052              ((NULL NL) (APPEND NODELIST NIL))
053              (OR (TMS-IS-IN (CAR NL)) (RETURN (LIST (CAR NL))))))
054
055   (DEFUN TMS-NWF-OUT-EXTRACT (NODELIST)
056          (DO ((NL NODELIST (CDR NL)))
057              ((NULL NL) (APPEND NODELIST NIL))
058              (AND (TMS-IS-IN (CAR NL)) (RETURN (LIST (CAR NL))))))
```

```
001
002  (COMMENT DEPENDENCY-DIRECTED BACKTRACKING SYSTEM)
003
004  ;;; TMS-CONTRADICTION IS THE FUNDAMENTAL METHOD FOR DECLARING A SET
005  ;;; A SET OF NODES CONTRADICTORY.  THE ARGUMENTS ARE A CONTRADICTION
006  ;;; TYPE, WHICH IS A MNEMONIC SYMBOL, A LIST OF NODES TO BE USED AS
007  ;;; THE SUPPORT OF THE CONTRADICTION, AND THE EXTERNAL ARGUMENT FOR
008  ;;; THE CONTRADICTION, AS IN TMS-SL-JUSTIFY.  THE FINAL ARGUMENT IS THE
009  ;;; CONTRADICTION FUNCTION TO BE CALLED WHEN NO ASSUMPTIONS CAN BE FOUND.
010
011  (DEFUN TMS-CONTRADICTION (CTYPE SUPPORT EXTARG CFUN)
012         (LET ((CONT (TMS-MAKE-FACT 'CONTRADICTION "(,CTYPE CONTRADICTION))))
013              (LET ((CNODE (TMS-FACT-NODE CONT)))
014                   (TMS-SL-JUSTIFY CNODE SUPPORT NIL EXTARG)
015                   (TMS-PROCESS-CONTRADICTION CONT CNODE CTYPE CFUN)
016                   CNODE)))
017
018  ;;; TMS-PROCESS-CONTRADICTION DIRECTS THE BACKTRACKING PROCESS.
019  ;;; ITS ARGUMENTS ARE THE CONTRADICTION NAME, THE CONTRADICTION NODE,
020  ;;; THE CONTRADICTION TYPE, AND THE CONTRADICTION FUNCTION.  IF NO
021  ;;; CONTRADICTION FUNCTION IS SUPPLIED, IT IS IGNORED.
022
023  ;;; THE THEORY OF BACKTRACKING IN THIS FUNCTION IS AS FOLLOWS:
024  ;;; THERE ARE 4 FLAVORS OF NODES AS FAR AS THE BACKTRACKER IS CONCERNED:
025  ;;; ASSUMPTIONS -- SUPERIORLESS IN NODES SUPPORTED BY OUT NODES.
026  ;;; SUSPECTS    -- OUT NODES SUPPORTING ASSUMPTIONS.
027  ;;; IN-SUPPORT  -- NODES WHICH ARE IN INDEPENDENT OF ANY SUSPECTS.
028  ;;; OUT-SUPPORT -- NODES WHICH ARE OUT INDEPENDENT OF ANY SUSPECTS.
029  ;;; BOTH TYPES OF INDEPENDENT SUPPORT ARE COLLECTED BY CALLING TMS-FINDINDEP
030  ;;; ON THE CONTRADICTION AND THE LIST OF ASSUMPTIONS.
031
032  ;;; *TMS-CONTRADICTION-ASSUMPTIONS* IS A LIST OF PAIRS OF ASSUMPTIONS AND THEIR SUSPECTS.
033
034  (DEFUN TMS-PROCESS-CONTRADICTION (CONT CNODE CTYPE CFUN)
035         (LET ((*TMS-NOTED-IN-NODES* NIL)
036               (*TMS-NOTED-OUT-NODES* NIL))
037              (TMS-PROCESS-CONTRADICTION1 CONT CNODE CTYPE CFUN)
038              (TMS-TMP-SCAN)
039              (TMS-SIGNAL-CHANGES)))
040
041  (DEFUN TMS-PROCESS-CONTRADICTION1 (CONT CNODE CTYPE CFUN)
042         (MAKE (TMS-CONTRADICTION-MARK CNODE) T)
043         (MAKE (TMS-CONTRADICTION-NAME CNODE) CONT)
044         (MAKE (TMS-CONTRADICTION-TYPE CNODE) CTYPE)
045         (AND CFUN (MAKE (TMS-CONTRADICTION-FUNCTION CNODE) CFUN))
046         (AND (TMS-IS-IN CNODE)
047              (LET ((*TMS-CONTRADICTION-ASSUMPTIONS* NIL))
048                   (COND ((TMS-FINDCHOICES CNODE)
049                          (COND (*TMS-SEE-CONTRADICTIONS-SW*    ;;; NOTIFY USER OF CONTRADICTION?
050                                 (TERPRI)
051                                 (PRINC '|CONTRADICTION: |)
052                                 (PRIN1 CONT)
053                                 (PRINC '| |)
054                                 (PRIN1 CTYPE)
055                                (COND (*TMS-SEE-CULPRITS-SW*   ;;; PRINT A LIST OF ALL ASSUMPTIONS?
056                                       (TERPRI)
057                                       (PRINC '|SUSPECTS: |)
058                                       (MAPC '(LAMBDA (N) (TMS-PRINT '|    | (CAR N)))
059                                             *TMS-CONTRADICTION-ASSUMPTIONS*)
060                                       (TERPRI))
061                                      (T (PRINC '| |)))
062                                (PRINC '|CULPRIT: |)
063                                (PRIN1 (TMS-EXTERNAL-NAME
064                                        (CAAR *TMS-CONTRADICTION-ASSUMPTIONS*)))
065                                (TERPRI)))
066
067                           ;;; THE FOLLOWING WILL CAUSE TRUTH MAINTENANCE, AND WILL RESULT
068                           ;;; IN ONE OF THE ASSUMPTIONS OR CHOICES BEING CHANGED
069                           (TMS-CONTRADICTION-ASSERT-NOGOOD CNODE)
070                           'FOUND-A-CULPRIT)
071
072                          ;;; IF THERE WERE NO UNDERLYING ASSUMPTIONS FOUND,
073                          ;;; CALL THE USER'S CONTRADICTION HANDLING FUNCTION IF IT EXISTS.
074                          (T (AND (TMS-CONTRADICTION-FUNCTION CNODE)
075                                  (FUNCALL (TMS-CONTRADICTION-FUNCTION CNODE)
076                                           (TMS-EXTERNAL-NAME CNODE)))
077                             'FOUND-NO-CHOICES)))))
```

```
001
002   ;;; CONTRADICTION-ASSERT-NOGOOD MAKES UP THE NOGOOD FOR A GIVEN CONTRADICTION.
003
004   ;;; NOGOODS ARE IMPLEMENTED AS DEPENDENCY RELATIONSHIPS, IN CONTRAST TO ARS,
005   ;;; SO THAT NO EXPLICT NOGOOD CHECKING IS NECESSARY - THE TRUTH MAINTENANCE
006   ;;; SYSTEM PERFORMS THAT TASK.  THUS IT IS KNOWN THAT NO FORMER SUSPECTS
007   ;;; THAT ARE OUT AT THE TIME OF A CONTRADICTION (OR ANY OTHER TIME)
008   ;;; ARE IN CONFLICT WITH A NOGOOD SET, FOR A NOGOOD SET WOULD CAUSE A
009   ;;; SUSPECT NODE TO BE IN.
010
011   ;;; NOTE THAT OUTSIDE OF THE NOGOOD ITSELF, ONLY ONE NEW JUSTIFICATION IS
012   ;;; CREATED FOR OUT NODES SUPPORTING ASSUMPTIONS.  THE NEW JUSTIFICATION
013   ;;; WILL NULLIFY THE CURRENT JUSTIFICATION OF THE FIRST ASSUMPTION ON THE LIST.
014   ;;; IF THAT ASSUMPTION HAS NO OTHER VALID SUPPORT, IT WILL GO OUT AND THE CONTRADICTION
015   ;;; SHOULD GO OUT AS WELL.  IF THE ASSUMPTION IS STILL IN DUE TO OTHER SUPPORT,
016   ;;; OR IF THE CONTRADICTION HAS OTHER SUPPORT, BACKTRACKING WILL BE INVOKED
017   ;;; AGAIN BY THE TMS-TMP-SCAN LOOP.  THUS OTHER NEW JUSTIFICATIONS MAY BE ADDED
018   ;;; LATER FOR THE OTHER ASSUMPTIONS SUPPORTING THE CONTRADICTION.
019
020   (DEFUN TMS-CONTRADICTION-ASSERT-NOGOOD (CNODE)
021          (LET ((AL (MAPCAR 'CAR *TMS-CONTRADICTION-ASSUMPTIONS*)))
022               (LET ((NOGOODF
023                      (TMS-MAKE-FACT
024                       'NOGOOD
025                       "(NOGOOD . ,(MAPCAR '(LAMBDA (N) (TMS-EXTERNAL-NAME N)) AL)))))
026                    (LET ((NOGOOD (TMS-FACT-NODE NOGOODF)))
027                         (MAKE (TMS-NOGOOD-CONTRADICTION NOGOOD) CNODE)
028                         (MAKE (TMS-NOGOOD-ASSUMPTIONS NOGOOD) AL)
029                         (MAKE (TMS-CONTRADICTION-NOGOODS CNODE)
030                               (CONS NOGOOD (TMS-CONTRADICTION-NOGOODS CNODE)))
031   ;;;CP
032   ;;; THIS COULD BE REPLACED BY A CALL TO TMS-FINDINDEP TO CONSTRUCT A NEW JUSTIFICATION.
033                         (TMS-CP-JUSTIFY1 NOGOOD CNODE AL NIL
034                                          "(NOGOOD FOR ,(TMS-EXTERNAL-NAME CNODE)))
035                         (LET ((P (CAR *TMS-CONTRADICTION-ASSUMPTIONS))
036                               (A (CAR AL)))
037                              (TMS-SL-JUSTIFY1 (CADR P)
038                                               "(,NOGOOD @(DELQ A (APPEND AL NIL)))
039                                               (CDDR P)
040                                               "(CULPRIT ,NOGOODF)))))))
```

```
001
002   ;;; TMS-FINDCHOICES MARKS THE SUPPORT OF A CONTRADICTION TO FIND THE RELEVANT CHOICES.
003   ;;; IT TRIES TO RETURN THE MAXIMAL ASSUMPTIONS.  THESE ARE THE ASSUMPTIONS INVOLVED
004   ;;; IN THE WELL-FOUNDED SUPPORT OF THE CONTRADICTION ON WHICH NO OTHER ASSUMPTIONS
005   ;;; IN THE WELL-FOUNDED SUPPORT DEPEND.  THIS IS TO AVOID THROWING AWAY MORE INFORMATION
006   ;;; THAN IS NECESSARY.  ALSO, IT IS PRAGMATICALLY USEFUL, SINCE THERE MAY NOT BE ENOUGH
007   ;;; INFORMATION TO LOGICALLY RULE OUT A NON-MAXIMAL ASSUMPTION.  THE RULING OUT OF THE
008   ;;; ASSUMPTION MUST INVOLVE THE NODE THAT THE OTHER ASSUMTIONS ARE STILL IN: BUT IF THEY
009   ;;; ARE SUPPORTED BY THE RULED-OUT ASSUMPTION, THEY WILL GO OUT AND THE ASSUMPTION WILL
010   ;;; NO LONGER BE RULED OUT.
011
012   ;;; THE TMS-SUPERIORS-MARK OF A NODE IS
013   ;;;      'YES  IF SOME CHOICE DEPENDS ON THE NODE
014   ;;;      'NO   IF NO CHOICE DEPENDS ON THE NODE
015   ;;;      NIL   IF THE NODE HAS NOT BEEN MARKED YET.
016
017   (DEFUN TMS-FINDCHOICES (NODE)
018          (TMS-FINDCHOICES1 NODE NIL)
019
020          ;;; THE FOLLOWING WEEDS OUT ANY SUBORDINATE ASSUMPTIONS SPURIOUSLY INCLUDED
021          ;;; IN THE LIST DUE TO CHRONOLOGICAL ACCIDENTS IN TMS-FINDCHOICES1
022          (SETQ *TMS-CONTRADICTION-ASSUMPTIONS*
023                (MAPCAN '(LAMBDA (S) (AND (EQ (TMS-SUPERIORS-MARK (CAR S)) 'NO) (LIST S)))
024                        *TMS-CONTRADICTION-ASSUMPTIONS*))
025
026          ;;;CLEANUP THE MARKS MADE BY TMS-FINDCHOICES1
027          (TMS-FINDCHOICES2 NODE)
028
029          ;;;RETURN AN INDICATION OF WHETHER THERE WERE ANY ASSUMPTIONS.
030          (NOT (NULL *TMS-CONTRADICTION-ASSUMPTIONS*)))
031
032   (DEFUN TMS-FINDCHOICES1 (NODE SUPERIORSP)
033          ;;; SUPERIORSP IS WHETHER THE NODE HAS CHOICES ABOVE
034          (EQCASE (TMS-SUPERIORS-MARK NODE)
035                  (YES)
036                  (NO (COND (SUPERIORSP                ;;; REVISIONS MUST BE PROPAGATED DOWNWARDS.
037                             (MAKE (TMS-SUPERIORS-MARK NODE) 'YES)
038                             (MAPC '(LAMBDA (A) (TMS-FINDCHOICES1 A SUPERIORSP))
039                                   (TMS-ANTECEDENTS NODE)))))
040                  (ELSE
041                   (COND (SUPERIORSP (MAKE (TMS-SUPERIORS-MARK NODE) 'YES))
042                         (T (MAKE (TMS-SUPERIORS-MARK NODE) 'NO)))
043                   (LET ((OUT-SUPPORT (MAPCAN '(LAMBDA (N) (AND (TMS-IS-OUT N) (LIST N)))
044                                              (TMS-ANTECEDENTS NODE))))
045                        (AND OUT-SUPPORT
046                             (COND ((NOT SUPERIORSP)
047                                    ;;; ONLY SUPERIORLESS CHOICES ARE COLLECTED.
048                                    ;;; THIS MAY COLLECT ASSUMPTIONS WHICH ARE SUBORDINATE.
049                                    (SETQ SUPERIORSP T)
050                                    (PUSH (CONS NODE OUT-SUPPORT)
051                                          *TMS-CONTRADICTION-ASSUMPTIONS*))))
052                             (MAPC '(LAMBDA (A) (TMS-FINDCHOICES1 A SUPERIORSP))
053                                   (TMS-ANTECEDENTS NODE))))))
054
055   (DEFUN TMS-FINDCHOICES2 (NODE)
056          (COND ((TMS-SUPERIORS-MARK NODE)
057                 (MAKE (TMS-SUPERIORS-MARK NODE) NIL)
058                 (MAPC 'TMS-FINDCHOICES2 (TMS-ANTECEDENTS NODE)))))
```

CS-TR Scanning Project
Document Control Form

Date : 3/21/96

Report # AI-TR-419

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 126(132-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other:_____

Check each if included with document:
☒ DOD Form ☒ Funding Agent Form ☒ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP!	(1-126)
UNNUMBERED TITLE PAGE, UNNUMBERED FUNDING AGENT, 2-97, APPENDIX #3	1-28
SCANCONTROL, COVER, DOD, TARGETS(3)	(127-132)

Scanning Agent Signoff:
Date Received: 3/21/96 Date Scanned: 4/1/96 Date Returned: 4/4/96

Scanning Agent Signature: Michael W. Cook

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
AI-TR-419

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE *(and Subtitle)*
Truth Maintenance Systems for Problem Solving

5. TYPE OF REPORT & PERIOD COVERED
Technical Report

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Jon Doyle

8. CONTRACT OR GRANT NUMBER(s)
N00014-75-C-0643

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetts 02139

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, Virginia 22209

12. REPORT DATE
January 1978

13. NUMBER OF PAGES
134

14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*
Office of Naval Research
Information Systems
Arlington, Virginia 22217

15. SECURITY CLASS. *(of this report)*
UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Artificial Intelligence
Problem Solving
Truth Maintenance
Dependencies
Logic
Backtracking
Explanation
Hierarchy

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The thesis developed here is that reasoning programs which take care to record the logical justifications for program beliefs can apply several powerful, but simple, domain-independent algorithms to 1) maintain the consistency of program beliefs, 2) realize substantial search efficiencies, and 3) automatically summarize explanations of program beliefs. This report describes techniques for representing, recording, maintaining, and using justifications for beliefs. Also presented is an annotated implementation of a domain-independent program.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**